

# The display of quadtree encoded pictures.

Milford, D. J.

*Award date:*
1984

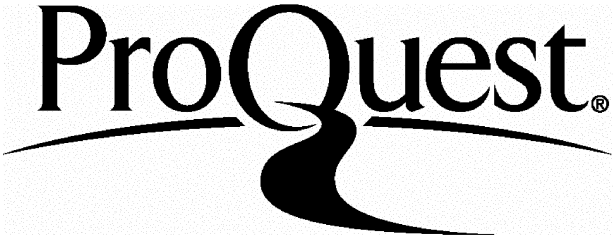
*Awarding institution:*
University of Bath

# THE DISPLAY OF QUADTREE ENCODED PICTURES

submitted by D. J. Milford
for the degree of Ph.D.
of the University of Bath

1984





ACKNOWLEDGEMENTS

I gratefully acknowledge the valuable help provided by various members of the University during the preparation of this thesis. Particular thanks are due to my supervisors Drs. P J Willis and J R Woodwark for their continual encouragement and guidance.

The pictures used in the thesis were generated using a volume modeller developed in the School of Engineering by Dr. J R Woodwark and others including Mr. A Wallis who helped transport the picture data.

I must also thank Dr. J B Hanson for assistance with software problems and Mr. K Simpkin whose careful hardware construction was greatly appreciated.

The work was financed by the Science and Engineering Research Council and I am pleased to acknowledge their support.

## SUMMARY

The data used by a computer to represent and display a picture conventionally requires very large amounts of dedicated storage. As the trend continues towards high resolution graphics employing a wide range of colours the associated volume of data often expands to a point where sophisticated techniques become essential to maintain a practical interactive system.

Considerable research has been conducted into coding schemes which compress the volume of picture data. These methods are clearly of great value in reducing the delay which results from transmitting picture data between sites. Codes which also serve directly as a source for real-time display are better still.

The quadtree method of picture encoding, which has previously been successfully employed in the field of picture processing, is shown in this thesis to provide the basis for an integrated approach to data compression in the field of computer graphics. The display of colour pictures from quadtree code is performed by customised hardware which operates in conjunction with a general purpose processor. A prototype system is described which provides efficient downloading of pictures from a host and rapid display of locally stored pictures. The amenability of quadtree code to manipulation is demonstrated by incorporating a limited capability to pan and zoom on a picture. Application of the method to picture archive retrieval is examined.

CONTENTS

# Chapter 1 INTRODUCTION

Rapid development in the field of micro-electronic engineering has resulted in a proliferation of computing machinery applied to an increasing variety of tasks. Moreover, the separation of user and machine which prevailed in the days of batch processing by large mainframe computers is fast disappearing with the emergence of powerful microprocessor-based systems. These small systems have not only decentralised computing facilities but have tended to focus attention upon the nature of the man-machine interface. Investigation of novel means for interacting with computers has become a major branch of computer science embracing such techniques as pattern recognition, voice recognition and speech synthesis. A now well established example of this trend is the use of computer generated graphics output.

## 1.1 Computer graphics technology

Throughout the development of computer graphics the cathode ray tube (CRT) has played a central role as the display device. The earliest systems required fairly elaborate electronics in order to trace lines on the screen by direct control of the beam deflection. These so-called calligraphic or refresh stroke displays are driven by repeated real-time conversion of a display list of line coordinates into a sequence of beam deflection signals. Consequently, if the drawing becomes too complex the scan repetition rate is reduced and flicker becomes annoying. The Penetron Tube adds a limited colour capability to calligraphic

displays by employing two layers of screen phosphor. The output colour can then be altered by modulating the beam energy. Refresh stroke displays provide high resolution images and still find a limited application where high quality and/or real-time animation of line drawings is required. However, their high cost and inflexibility precludes general use.

The introduction of storage tube displays in the early 1970s marked the advent of widely available, low cost computer graphics. Employing the same random deflection technology as the calligraphic displays, the need for repeated conversion of a display list is removed by accumulating line data on a screen which remains fluorescent indefinitely wherever the beam impinges upon it. The major disadvantage resulting from this technique is that local picture erasure is impossible.

While storage tube displays became increasingly used in a wide variety of applications their limitations in fields such as image processing encouraged the development of an alternative display technique. Based on the principles used in television picture generation, raster scan displays build an image as an array of picture element dots (pixels) scanned line by line across the screen. Most commonly the picture information is stored pixel by pixel in a dedicated area of memory known as a framestore. This provides three important advantages over random deflection display types. Firstly, images can be built and modified at the pixel level allowing exceptional scope in software. Secondly, the imposition of regular scanning brings all the benefits associated with television

technology including the ability to display areas. Thirdly, colour is available. Initially, raster scan displays were limited by the expense of framestore memory and by CRT technology to produce images whose resolution rarely exceeded TV quality. However, by the late 1970s the cost of semiconductor memory had become sufficiently small to make raster scan graphics using framestores a feasible proposition in many applications. Together with modern CRTs, which may permit as many as 1500 raster lines per frame, the use of raster scan techniques for high resolution display applications is now common. An additional boost to the framestore refreshed display was introduced by the advent of low cost microprocessors dedicated to manipulating pixel values in the framestore thus removing this burden from the main system processor. The integration of raster scan displays into stand-alone and networked computer workstations is a logical consequence of this development.

Whilst CRT displays dominate the field of computer graphics there are several other types which find a limited application, most notably the plasma display. Plasma panel displays comprise what is effectively an array of gas discharge cells. They are preferred to CRTs in applications where restricted space and robustness are major considerations but are more difficult to interface and generally limited to a single display colour.

### 1.1.1 Interactive peripherals

The graphics display is essentially an output device for communicating information to the user. However, in many applications the display can also provide a useful format for the

input of information to the system and to this end a variety of peripherals have been developed. The most common means of interacting with a display is through a screen cursor which may be positioned by the user manipulating a joystick, mouse, tracker ball, light pen or bit pad sensor. Having set the cursor at the required position on the display a command may be issued from a standard keyboard or customised button box. Interaction of this kind is increasingly common in menu-driven computer systems where unsophisticated graphics is used to create a "user-friendly" interface to the operator. Similar techniques also apply to true graphics applications such as drawing systems which allow real-time modification of the displayed image. An even more immediate (though not particularly precise) means of interacting with a display is provided by the touch sensitive screen which simply requires the user to point at the relevant spot on the screen.

## 1.2 Computer graphics applications

In common with many technologies, the growth of applications for computer graphics has followed the development and availability of hardware. Nowadays graphics are widely used in business and industrial applications to present information which would previously have been issued in the form of tables of numerical data. Operators of process and plant control systems are provided with an intelligible overview by graphic monitoring if it is well organised and suitably colour coded. In many control applications such as power station monitoring and air traffic control the systems have become so complex that graphical output must now be

considered indispensible.

Perhaps the most interesting range of applications are those which, without computer graphics, would never have evolved. Computer aided design (CAD) developed in the late 1960s and was initially applied to mechanical engineering problems in the automobile and aerospace industries. Using interactive graphics facilities designers are able to model components and run appropriate test procedures with rapidly available results. A natural development of CAD is computer aided manufacture (CAM) which controls production of components designed by CAD. More recently CAD/CAM has been introduced to the electronics industry, initially for printed circuit design and lately for the design of very large scale integrated circuits (VLSI).

Computers in general, and graphics in particular, have had a profound influence on the printing industry in recent years. Character font design and page layout are now largely computer assisted tasks.

The entertainments and leisure industry has been quick to realise the potential of computer graphics. Ranging from the (often crude) graphics employed in arcade games to the refined techniques afforded by TV advertising and feature films such as "Tron" some strikingly novel visual effects have been produced.

Probably the most challenging problem confronting computer graphics at present is real-time visual simulation. There are clearly many advantages to be gained by using a simulator for training personnel engaged in hazardous occupations. Ideally the

trainee should obtain an entirely realistic experience during simulation and the visual aspect is often the most vital. Unfortunately the complexity of graphics which can presently be generated in an interactive real-time environment is rather limited by the excessive amount of computation involved. Several systems have been produced which successfully provide night-time flight simulation and, since the rate of movement is slower, systems which simulate views from a ship's bridge are also practicable. Daytime simulators are also available but these demand extensive special purpose hardware and are therefore very costly.

### 1.2.1 Graphics software

Not surprisingly, the booming market in graphics hardware has generated its own problems. Through lack of standardisation the development of graphics software has been severely limited by the need for packages customised to individual machines. Although this has, for a considerable time, been recognised by users and manufacturers alike as a serious problem little progress on standardisation was made until recently. Attempts in the USA to promote the ACM CORE standard seem to have failed and the European Graphics Kernel System (GKS) now appears to offer the best hope for a universally accepted standard.

## 1.3 The thesis

Raster scan displays refreshed from a framestore have become firmly established as the norm for computer graphics systems. The hardware is now available to produce high resolution graphics with

full tonal colour rendition at a moderate cost. There are. furthermore, persuasive software arguments for full frame storage. A pixel by pixel representation provides picture data in a form which is easy for a programmer to access and manipulate in a most general way. Unfortunately there is a severe penalty to be paid for this convenience; the size of framestore required to represent a high resolution colour image is likely to be a megabyte or more. While this may not prove to be a problem in cost terms it certainly presents an obstacle to fast system response in many instances. Consider, for example, the problem of picture storage and retrieval when such large amounts of data are involved. If the mass storage unit is closely coupled to the display then data can be downloaded to the framestore using a fast DMA interface. In the case of remote storage the bandwidth of conventional communication channels limits the rate of data transfer to a degree where interactive response becomes slow. Indeed, with users accustomed to the increasingly rapid response of computing systems in general, excessively slow graphics response may be unacceptable.

This thesis addresses the problem of picture display from remote digital storage with particular reference to the interactive demands normally associated with computer graphics applications.

Chapter 2 discusses a variety of techniques which may used to compress the large amount of data associated with framestore display. The concept of picture representation and its relationship to coding in the context of computer graphics is also examined.

Chapter 3 focusses on the quadtree as a suitable structure for picture representation. A number of coding variations are examined.

Chapter 4 describes the development of a scheme by which pictures may be generated in real-time using a form of quadtree code as the display source.

Chapter 5 describes the realisation in hardware of the principles outlined in the previous chapter. Operation of the more crucial parts of the circuitry is described in detail.

Chapter 6 presents details of a prototype picture system which was developed to examine and quantify the characteristics of quadtree encoding for picture retrieval and display. The system includes basic facilities to pan and zoom on a displayed picture.

Chapter 7 contains information on system performance. Statistics are included which provide a measure of the data compression achieved by quadtree encoding and enable prediction of quadtree growth as picture resolution is increased.

Chapter 8 outlines proposals for two applications in which quadtree encoding of pictures is likely to prove practical and beneficial. The potential of quadtree picture code for mainstream interactive graphics applications is also briefly examined.

# Chapter 2 PICTURE REPRESENTATION AND CODING

From the previous chapter it is clear that the range of application for computer graphics is large and, at present, rapidly expanding. It is perhaps surprising, therefore, that relatively little innovation has occurred in the basic concepts underlying the design of raster-scan graphics displays. This chapter describes the conventional fully buffered frame store (bit-mapped) display approach and continues with a survey of some alternative strategies which have been proposed.

## 2.1 Digital representation of pictures

In mathematical terms a picture may be regarded as a function of two variables

$$P = f(x,y)$$

where x,y are bounded spatial coordinates in the picture plane. For a monochrome picture, P is a real valued measurement of the image intensity and is generally referred to as the grey level; for a colour picture P is a triplet of real values (see Appendix 2).

Formulation of a digital representation involves a notional division of the picture area into a regular array of equal sized elements (pixels) within each of which P is constant. The two dimensional picture function is thereby replaced by a matrix of pixel values which, in digital form, are necessarily quantised. Given a finite set of pixel values, these may constitute a code representing any desired combination of intensity and colour characteristic.

## 2.2 Bit-mapped displays

The conceptual representation described above is made a physical reality in a bit-mapped display system by assigning computer storage to each pixel in the picture frame. The memory thus used is frequently referred to as a frame buffer or framestore. This means, for example, that a picture defined to a resolution of 512 x 512 pixels requires 256K storage cells each with a capacity determined by the colour range. A picture containing, for instance, 256 colours or grey levels requires eight bits of storage per pixel. The hardware necessary to generate a display from this data is uncomplicated but the principal benefit of such a scheme lies in its suitability for interactive applications. With relatively simple interfacing and software it is possible for a user to compose or modify a displayed picture. Automatic generation of primitive geometric forms is possible with very little computation owing to the simple correspondence which can be effected between display coordinates and data storage addressing.

Despite the obvious advantages of bit mapping this approach suffers from one major drawback, namely the large quantity of data required to represent a medium or high resolution picture with a large colour range. Furthermore, the situation worsens disproportionately as higher and higher resolutions are demanded; a doubling in spatial resolution entails a fourfold increase in the amount of data needed to represent the picture. With semiconductor memory continuing to become cheaper the cost factor in primary

display storage is not so important as it once was, although the quantity of secondary storage required for keeping picture files remains a problem. But by far the most serious consequence of using such an extensive picture representation is the time consumed in filling the frame buffer with pixel data when the display needs to be completely updated. The delay becomes all the more unacceptable when data has to be fetched from remote storage.

## 2.3 Pictorial data compression

Information theory provides a useful basis for any data compression scheme (see, for example, Hall [1] pp.312-325). Consider, as before, a picture defined to a resolution of 512 x 512 pixels with each pixel value stored as an eight bit code. The number of "pictures" which can be realised in this representation is therefore 256 raised to the power of 512 x 512. Of these, only a fraction are pictures of practical significance. In computer graphics, for instance, the image commonly contains substantial areas of uniform colour. The actual information conveyed by a meaningful picture is thus considerably less than the potential information which may be represented by a full pixel description. This information redundancy implies that a picture can be coded so that, on average, fewer bits per pixel are required. Picture entropy is a common measure of information content and is generally expressed as the average number of bits per pixel required to represent a given image. The aim of picture coding is to devise a practical scheme by which the theoretical picture entropy can be realised.

## 2.4 Picture coding for television

To a large extent, investigation into picture coding has been motivated by the search for an efficient digital transmission technique for video signals [1-5]. The benefits of moving away from analogue methods include improved noise immunity and simplified conversion between different TV standards. In the case of digital speech encoding it has been possible to transmit intelligibly over surprisingly narrow bandwidths; attempts at picture coding, have, as we shall see, proved less rewarding. It is generally accepted that by employing eight bit quantisation and a sampling rate of around 13 MHz a video signal can be digitised and used to reconstruct a picture of more or less TV quality. Real-time transmission of this data consequently requires a rate of more than 100 Mbits per second which, using pulse code modulation, is equivalent to a bandwidth of 100 MHz. Compare this with the UK analogue bandwidth of 5.5 MHz and it is clear that substantial data compression is desirable. Any attempt to decrease the bandwidth by reducing either the sampling rate or number of quantisation levels leads to unacceptable deterioration of picture quality. Undersampling causes a phenomenon known as aliasing and coarse quantisation produces an effect called false contouring. These effects are nicely illustrated in Hall [1] pp.93,94,98.

TV picture coding techniques fall into two main categories: intraframe coding and interframe coding. The second of these exploits the similarities which exist between successive frames of a TV picture by transmitting only the differences. This class of

code will not be discussed further since it is irrelevant to the coding of a static image. Intraframe codes are of greater interest since they exploit coherence within the picture frame and are therefore, in the first instance, concerned with static images. These codes may be further divided into two classes termed reversible and non-reversible. Reversible encoding allows exact reconstruction of the pixel representation but appears to offer only modest compression ratios of about 2:1 (see Hall [1] p.311). A greater degree of code efficiency may be achieved by non-reversible codes which allow a certain amount of approximation beyond the initial sampling and quantisation. Study of the psychophysical properties of visual perception has resulted in a range of approximation techniques which do not substantially affect subjective picture quality [6].

## 2.5 Picture coding for computer graphics

Before proceeding to a survey of coding methods it is worth examining the significance of digital TV techniques in computer graphics applications. Superficially the requirements appear to be similar; a picture must be generated in real-time from encoded data. Some authors differentiate between "real" images and those generated by computer but this distinction seems not to be fundamental and is certainly outdated when one considers the quality of image which can be synthesised nowadays. Therefore, if picture display is the only consideration then digital TV encoding methods seem relevant. This, however, ignores a most vital aspect of computer graphics, namely its interactive nature. In computer

graphics the picture code must fulfil a role which is absent in TV applications: it must serve not only as a source for display generation but also as a representation of the displayed image. It is difficult to see how approximation coding can be applied in these circumstances since graphic entities, although visually well defined, are likely to be ill-defined in the data structure.

The following survey therefore includes only those TV techniques which preserve the original pixel description together with examples of coding which have previously been used in the computer generation of images.

### 2.5.1 Coding of line drawings

Where a picture consists of a limited number of line segments drawn against a uniform background it may be efficiently encoded by a minimal specification of coordinates and dimensions. A code which caters for straight lines, arcs of circles, rectangles and other simple geometric shapes is quite feasible and, in effect, provides an ordered set of drawing commands. Provision may also be made for infilling bounded areas created by previously defined lines. Clearly, a code of this kind has very limited graphics application which cannot include the portrayal of detailed shaded images.

### 2.5.2 Block coding

In block coding the picture is divided into rectangular cells of m by n pixels and a code is allocated to each of the relevant cell configurations. This is the scheme adopted in the majority of alphanumeric terminals where a cell of typically 5 x 7 or 7 x 9

pixels is used to create the character set. In this case data compression is achieved because the characters form a small subset of the possible cell configurations. The low resolution graphics facility contained in Teletext uses a 2 x 3 cell and sixty four codes are therefore needed to represent the entire graphics set. These together with an alphanumeric set and various controls constitute a 7 bit code. It is worth noting that in this case compression is only achieved because the system employs a range of eight colours and uses a single control code to prefix a run of uniform-colour graphics characters. In a monochromatic system this scheme would offer no compression of graphics data.

Jordan and Barrett [7] have described a display which builds line drawings using an extension of the block coding technique. They use a character generator which provides not only alphanumerics but also a set of 108 basic line segment patterns drawn within a cell of 8 x 8 pixels. Each display file instruction includes, with the pattern code, codes which allow translation, reflection and masking of the basic pattern within the cell. Combined with various other features this permits the construction of complicated figures from cell codes. The authors report 30% data compression at limiting picture complexity when 25% of the screen is filled with lines.

Further examples of block coding may be found in a recent review by Kunt and Johnsen [8].

### 2.5.3 Run-length coding

It is a consequence of the raster scan technique that picture coherence is most easily exploited in one dimension only. Run-length encoding achieves data compression by inspecting pixel to pixel correlation along each raster line, a run of pixels with constant colour being coded with a pair of descriptors giving colour and run length. Cherry et al. [9] report the generation of TV quality pictures using run-length code with an average 3 bits per pixel. A major advantage of this coding technique is that it may be implemented at low cost with relatively unsophisticated hardware [10]. A commercial example is the DEC VT31C [11].

### 2.5.4 Predictive differential quantisation (PDQ)

Used alone, PDQ is an approximation method which exploits the high correlation between neighbouring pixel intensities. However, used in conjunction with other schemes it may form part of an exact code so the principle is described below. By using some sort of averaging function it is possible to formulate a prediction of a pixel value in terms of its predecessors. The pixel can therefore be encoded as the quantised difference between the predicted and the actual value and decoded using the same rule. The bit rate for a picture is thereby reduced because the differences may be coded with fewer bits than the absolute values. Large difference signals are only encountered when significant discontinuities in the image are crossed so this strategy inevitably introduces approximation unless alternatives exist for this special case. In the simplest scheme, known as Delta Modulation, the difference is calculated

between the pixel under consideration and its immediate predecessor in the scan line. Individual pixel values may therefore be restored by a simple integrator. Alternatively, to exploit vertical coherence, differences between corresponding pixels on successive scan lines may be used, in which case a line buffer must be maintained.

Huang [12] has described a PDQ variant which serves as an extension of run-length encoding to two dimensions. Together with special codes which mark the start and end (in the vertical extent) of coherent areas, the picture is encoded as differences between the position and length of runs on successive lines.

In a similar but more elaborate scheme Daskalakis et al. [13,14] augment the technique of run-length encoding by the use of line-to-line correlation. They define a selection of special codewords which permit the efficient description of colour transitions which occur within three pixels of an identical transition on the previous scanline. This refinement is reported to give compression ratios which improve upon simple run-length code by a factor which varies roughly from two to five depending upon the picture type.

### 2.5.5 Area coding

While schemes which combine point-to-point with line-to-line correlation certainly exploit two dimensional image coherence they nevertheless differentiate between the dimensions in a quite artificial way. This is, of course, a reflection of the raster technique of image construction. Considering the logical

equivalence of the two dimensions in a picture there are strong arguments (2.6) for a strategy which codes areas explicitly.

Schreiber et al. [15] have explored this possibility with a contour coding technique which they apply to typescript and to continuous tone images. A contour is traced by specifying a starting point followed by a sequence of three bit words which act as incremental vectors. A compression ratio of only 1.6 is quoted for a picture using 16 grey levels defined to a resolution of 256 by 256 pixels.

Kutsuzawa et al. [16], reporting on their "Video Signal Generator" (VSG), describe a colour graphic display which assembles a picture from a number of polygonal areas defined by their boundaries. The display file consists of a list of records which define each edge in terms of its uppermost X and Y coordinates, its gradient, its lowermost Y coordinate and the colour lying to the right of the edge. Customised hardware performs scan-conversion on this data but is limited to processing no more than thirty two colour transitions per raster line.

Working along similar lines, researchers at the University of Sussex [17] have designed a "Zone Management Processor" (ZMP) for computer animated images. Their pictures are composed of "quadrilateral" zones with sides specified by first, second or third degree equations. Unlike the VSG, the ZMP processes data records which contain a complete specification of each zone. Although this duplicates boundary information it aids recognition of coherent picture areas. In practice several ZMPs are used to

process data in parallel and their number defines the complexity of the displayed image; the use of sixteen ZMPs limits the display to no more than sixteen zones intersected by any raster line.

### 2.5.6 Bit-plane coding

This technique transforms a picture with 2^n grey levels into n separate binary images. It is found in practice that for natural images there is considerable coherence in the binary image associated with the most significant bit. The complexity of the least significant bit plane image is greatest. Having transformed the original image in this way the individual binary images may be encoded using one of the methods already described. Spencer and Huang [18] report a compression ratio of roughly 2:1 for 6 bit images with a resolution of 256 x 256 pixels. They obtained optimum results by Gray-coding the original image and run-length encoding the binary components.

### 2.5.7 Statistical coding

Many of the coding methods which have been described in this section may attain greater efficiency by applying a statistical analysis to the code. The object is to reduce the average number of bits per codeword by replacing fixed length codes with variable length codes. A Huffman code [19] is optimised to represent frequently occurring codes with fewer bits than infrequent codes. Except for transmission purposes the advantages offered by such a scheme to interactive computer graphics are generally outweighed by the complications introduced by variable length codes.

## 2.6 Conclusion

It is clear that a variety of coding techniques are available to tackle the major problem of the amount of data required to generate a picture. In order to assess the relative merit of these techniques in the context of computer graphics the following considerations seem relevant.

a) Data compression must remain a primary objective, particularly with regard to picture transmission and storage. It is clearly sensible that the amount of data required to represent a picture corresponds to the information content of the picture. It appears that compression ratios of about 2:1 can be expected for natural images with considerably better ratios for stylised computer generated images. Furthermore, there is reason to believe that for a fixed number of quantisation levels the compression ratio achieved by encoding improves as spatial resolution increases [20].

b) A coding scheme cannot be considered fully satisfactory if it fails to accommodate a full range of picture types. Naturally the efficiency of any particular code is likely to depend upon the image it is applied to but that is another matter.

c) The full benefit of data compression is not seen unless the compressed code can act as a source for display generation. Of course, it is sensible to employ picture coding for efficient transmission and storage even when a picture is eventually displayed using a conventional framestore design. The ultimate goal, however, is to dispense with the need to buffer individual pixel data.

d) The dual role of picture code in the context of computer graphics has already been remarked upon (2.5) and requires elaboration here. As we have seen. much of the work on picture coding has been orientated towards finding efficient transmission formats in which the data structure is only tenuously related to the image structure as perceived by the viewer. Thus, the code is invariably application-specific and of limited general use. What is required in the context of computer graphics is a compression technique which preserves the essential structure of the picture because this aids interaction with the picture [21]. Suppose the user wishes to manipulate or modify a displayed image. Ideally this will be achieved by direct manipulation of the picture code but in practice it can only be done if the data is structured to form a suitable picture representation. If not, then the code must be expanded into a form suitable for processing and finally recompressed. The use of compressed code as a display source then becomes questionable.

The contention of this thesis is that the only satisfactory method of picture coding for general use must exploit area coherence in a direct rather than indirect fashion.

Chapter 3 PICTURES AS QUADTREES

It would appear that many of the techniques previously applied to the problem of digital picture representation are unsatisfactory in the context of computer graphics, not because they fail to achieve a substantial degree of data compression but because they generate data which corresponds only indirectly with the observable picture structure. Clearly then, there is good reason to move attention away from research directed at efficient transmission and display generation toward areas of study where the structured organisation of picture data is all-important. Workers in several fields have employed a representation based on the subdivision of a picture by regular decomposition. Knowlton [22] has proposed a scheme which progressively resolves the finer detail of a picture by recursive division of the picture area into halves. Decomposition into quarters is more common and results in a structure which is variously referred to as Warnock-type [23], pyramid-data [24], segmentation tree [25], decomposition tree [26] and quadtree [27-33].

## 3.1 The quadtree scheme

The quadtree is a heirarchical representation of a picture in which the picture area is divided recursively into quadrants until the sub-areas so formed are uniform in colour (fig. 3.1). The subdivision may proceed to a limit determined by the resolution of the device acquiring or displaying the image which, in the case of a raster scan display, corresponds to pixel size. A square picture

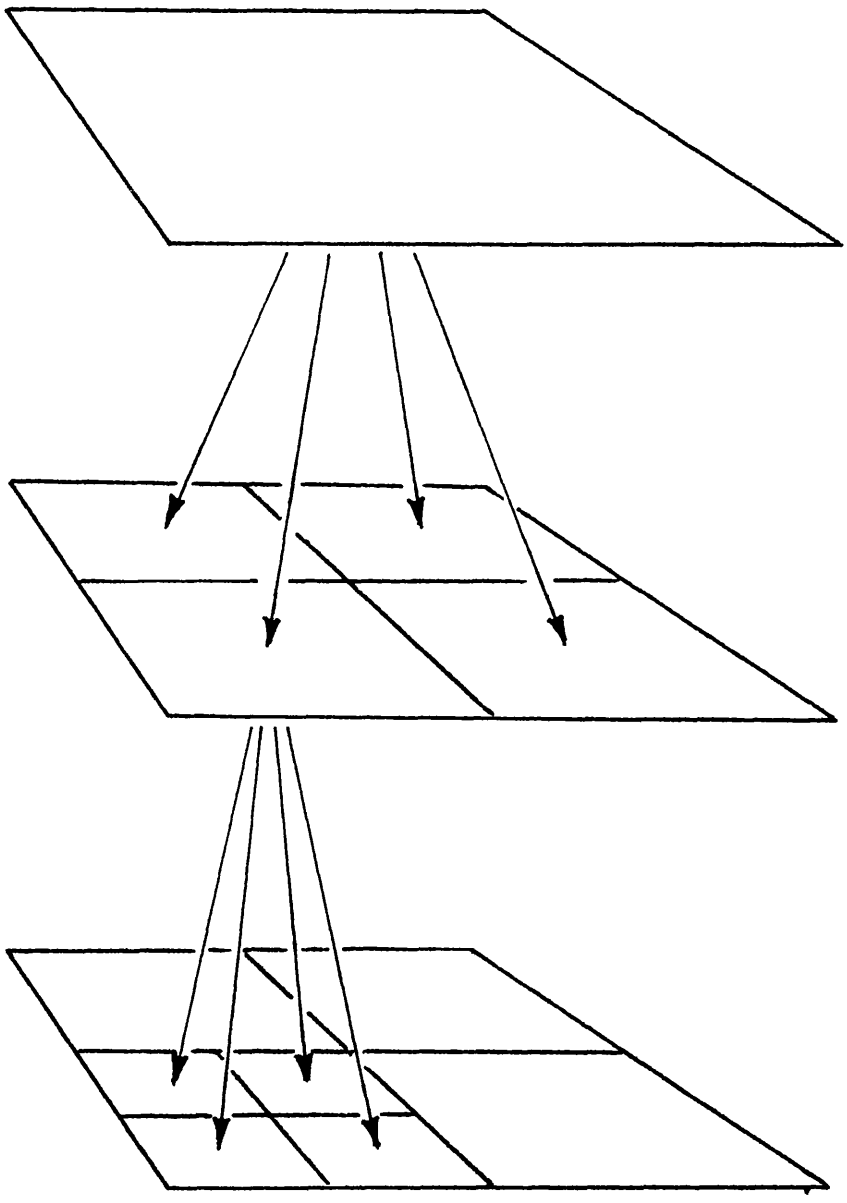


Figure 3.1 Recursive division of a picture

is therefore represented as a four-branched tree in which the nodes correspond to quadrants and the leaves correspond to square areas of uniform colour which will be referred to as quads (fig. 3.2). Although this appears to restrict the technique to dealing with displays configured as 2^n by 2^n pixels there are straightforward means of adapting the principle to other geometries (see the discussion on zoning in Chapter 6).

Interest within the image processing community derives from the fact that the quadtree provides an excellent storage structure which allows a picture to be represented at progressively finer levels of resolution. The quadtree exploits area coherence and at the same time preserves spatial information thus enabling the creation of efficient picture processing algorithms which can focus quickly upon areas of interest.

Given that it appears to serve as an excellent representation of pictures, the quadtree method also promises a reasonable degree of data compression [34,39]. Before considering its suitability from a display point of view it is appropriate to survey the various data structures which have been proposed for the storage of pictures as quadtrees.

## 3.2 Quadtree data structures

### 3.2.1 Linked tree structure

The quad tree representation of a picture is most naturally translated into a data structure which retains explicit links between tree node records. A minimal structure would consist of records with four fields describing the four descendants at each

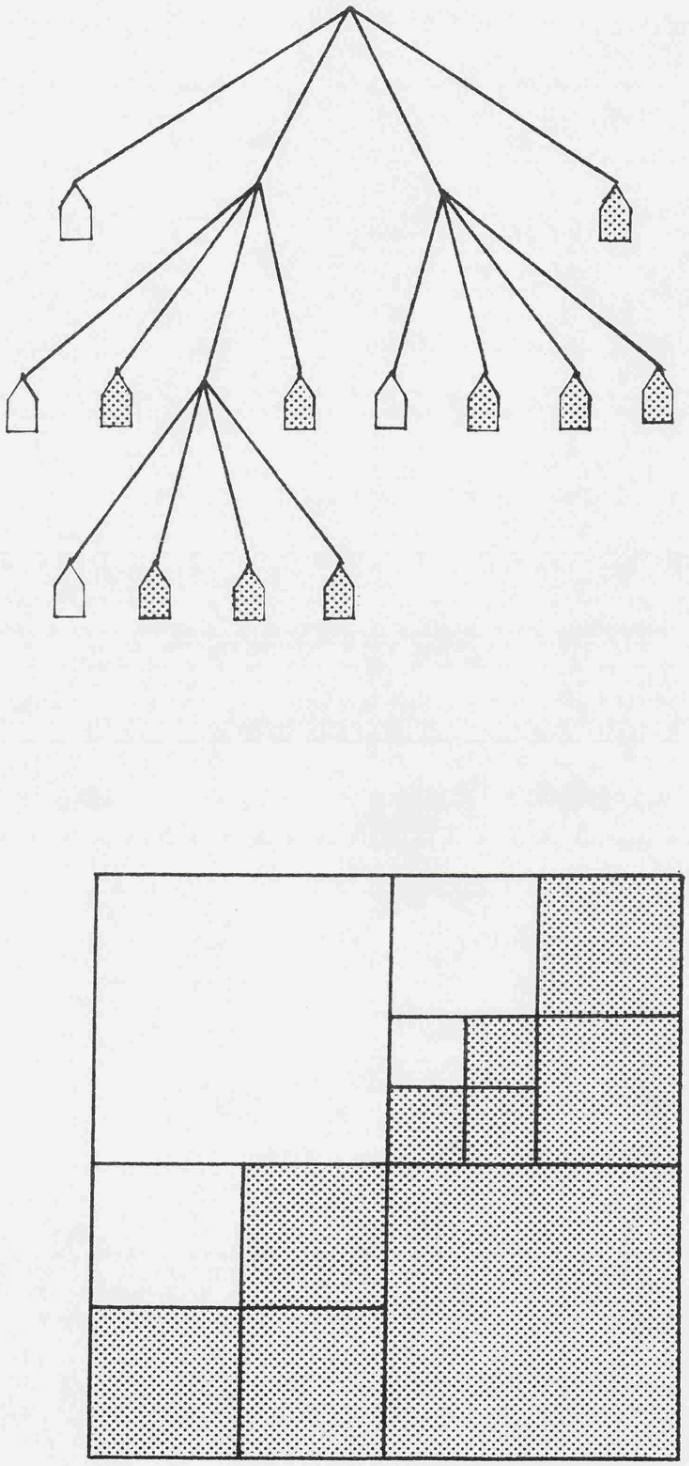


Figure 3.2 A quadtree and the corresponding picture subdivision

node: if the son is a leaf then the field contains colour data: if not then the field contains a pointer to the descendant node (fig. 3.3). This type of structure has been favoured for image processing because of its flexibility and it is sometimes embellished with backpointers to father nodes [30-33]. Hunter and Steiglitz [28] even incorporate "ropes" which provide links across the tree between adjacent nodes to assist identification of connected regions. Despite the advantages of a pointer structure there is a significant penalty in terms of storage space so data compression is diminished. Analysis of several picture trees (Chapter 7) indicates that the number of leaves is 75% of the total number of nodes. Therefore, even with the minimal requirement, pointers comprise 25% of the data. The problem is aggravated by the fact that for a reasonably complex picture the field width needed to store pointers is likely to equal or exceed 14 bits (spanning a 16K address space). Colour information is frequently coded with 8 bits or less.

### 3.2.2 Universal quadtree structure

Woodwark [35,36] has made considerable use of quadtrees in a volume modelling system. He has proposed a structure [37] in which every possible quadtree node is allocated storage space, thereby creating a very sparsely populated structure in which the linkage is implicit in the addressing scheme. Although this requires one third more storage than a full pixel representation, the structure may be rapidly traversed by applying simple Boolean operators to node addresses. Despite its extensive storage requirement the

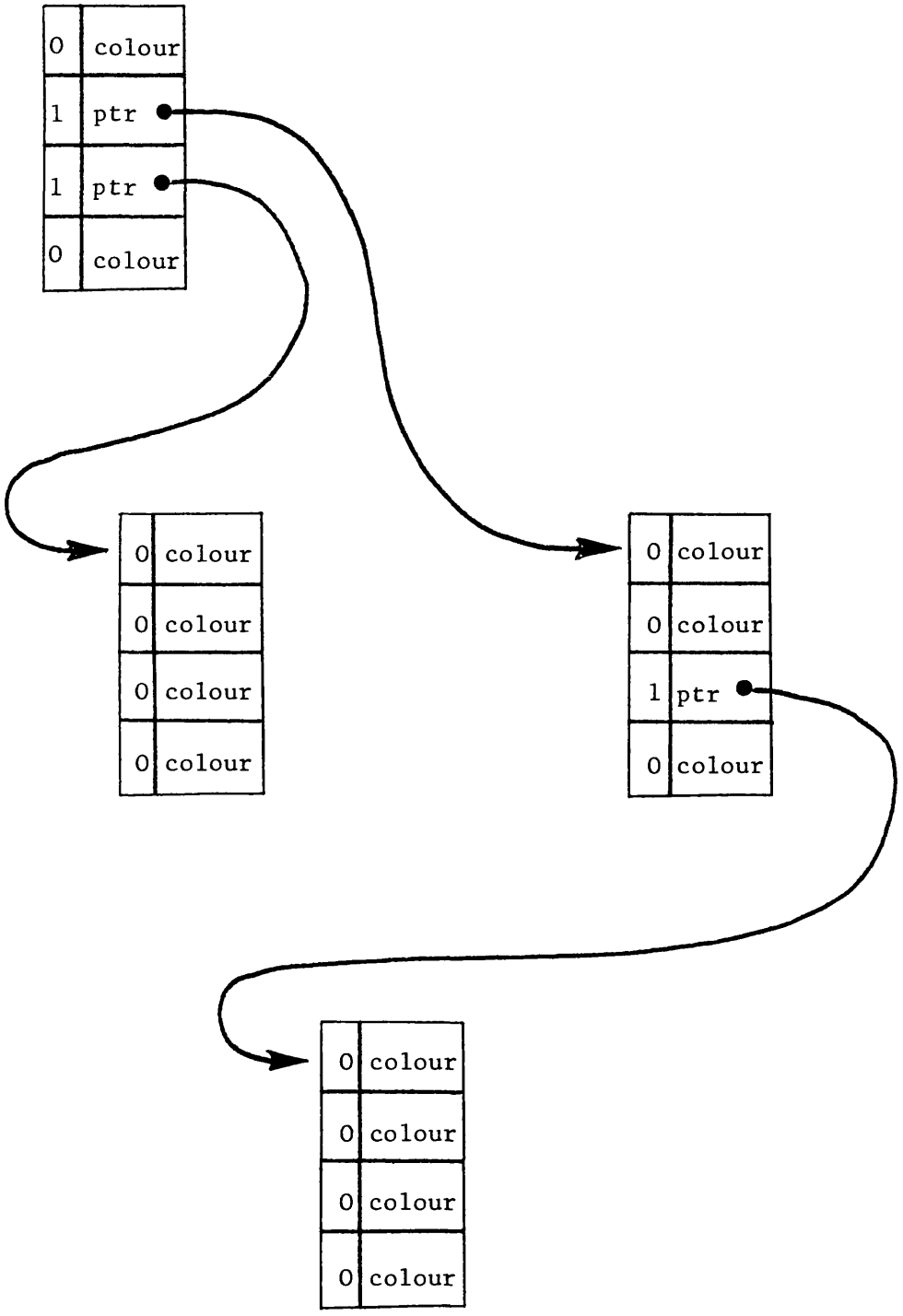



Figure 3.3 Linked tree data structure

structure nevertheless allows a picture to be represented with less data than the linked tree.

### 3.2.3 Implicit traversal

Given a quadtree structure there are several obvious ways in which it can be logically traversed to visit each node once and once only. Therefore it is possible to represent a quadtree without using pointers by a list of nodes in which the ordering is implicit.

Oliver and Wiseman [38] present a collection of picture manipulation algorithms which operate on a linear structure in which the nodes are ordered by a depth first traversal (fig. 3.4a). Each list record is marked by a single bit field as either a leaf or non-leaf node. The remainder of the record is used to code colour information: in the leaf case the code represents the quad colour; in the non-leaf case the code represents the "averaged" value of the sub-tree quads (figs. 3.4b, 3.4c). This enables the construction of a picture at various resolutions with anti-aliasing as a built in feature. Whilst averaging may reduce to simple arithmetic with a grey-scale picture it poses a considerable problem for coloured pictures generated via a look-up table (see Appendix 2) and may prove impossible where a small colour range is available.

Kawaguchi and Endo [34] employ a similar strategy in their representation of binary pictures. Their linear structure (called a DF-expression) employs three symbols ordered by a depth first traversal of the picture quadtree. Using "0" to represent white and

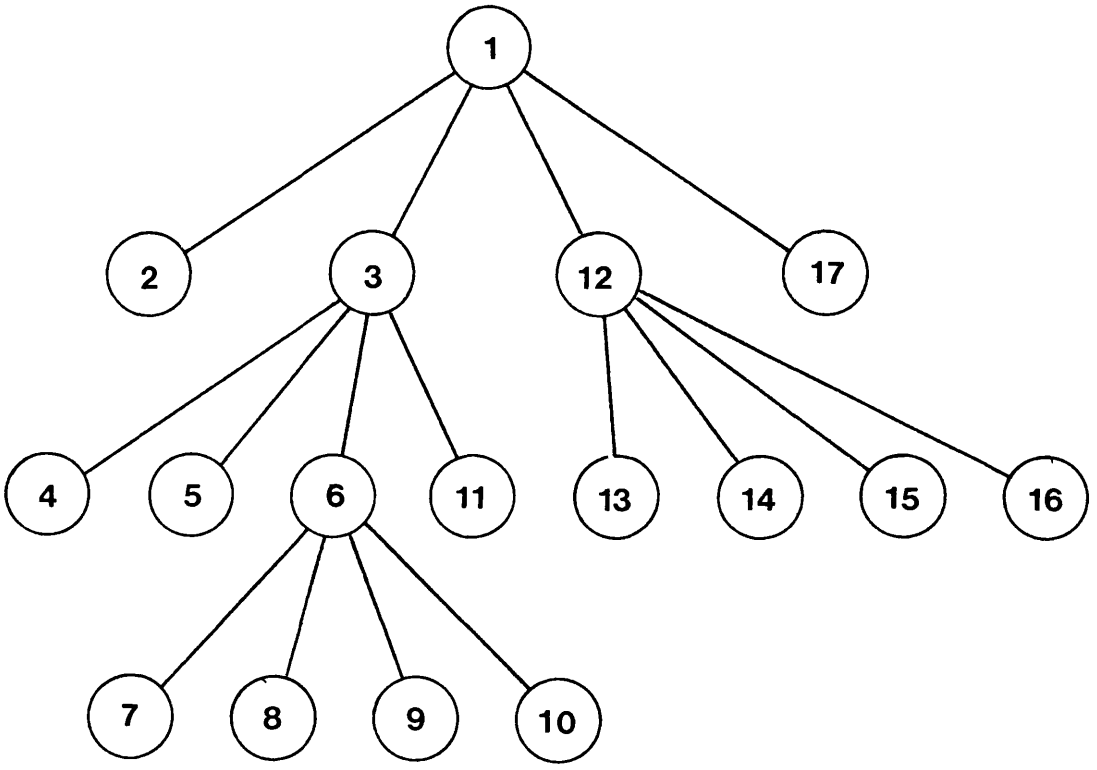



Figure 3.4a The order of nodes in a depth first traversal

the numbers shown represent the grey-scale quad intensities

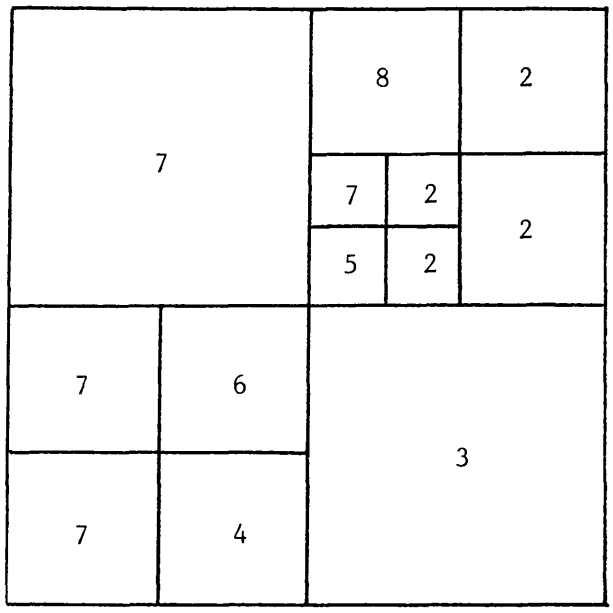



Figure 3.4b Picture subdivision corresponding to fig. 3.4a

| | | |
|---|---|---|
| 1 | 5 | picture average |
| 0 | 7 | NW QUADRANT |
| 1 | 4 | NE QUADRANT |
| 0 | 8 | |
| 0 | 2 | |
| 1 | 4 | |
| 0 | 7 | |
| 0 | 2 | |
| 0 | 5 | |
| 0 | 2 | |
| 0 | 2 | |
| 1 | 6 | SW QUADRANT |
| 0 | 7 | |
| 0 | 6 | |
| 0 | 7 | |
| 0 | 4 | |
| 0 | 3 | SE QUADRANT |

Figure 3.4c Depth first linear data structure

"1" to represent black the sequence of symbols

((00101(101(1100(0001(1010(11000

provides an unambiguous representation of a sub-tree where the symbol "(" announces the imminent subdivision of a node. The completion of each node is easily inferred: to clarify the issue the sequence is repeated below using redundant ")" symbols.

((0010)1(101(1100))(0001)(1010)(1100)0)

The authors evaluated three coding schemes for representing their three symbol alphabet and compared the results with other compression methods applied to identical pictures. They report compression factors more than 2.5 times that obtained using run-length code and about 1.5 times that using a predictive code.

Woodwark [39] has proposed a modification to the scheme of Oliver and Wiseman (above) which replaces the averages held at non-leaf nodes by special codewords which describe the configuration of descendants from the node. The number of colour codes is reduced wherever two or more of the descendants are leaves with the same colour. Comparison with run-length code for identical pictures suggests that this compression method is very nearly as efficient in terms of storage. Unlike run-length code it provides a database which permits a variety of picture manipulations.

Breadth first traversal (fig. 3.5) appears promising if averages are held because reduced resolution images can then be constructed from a linear sub-list [38]. In other respects it seems less useful.

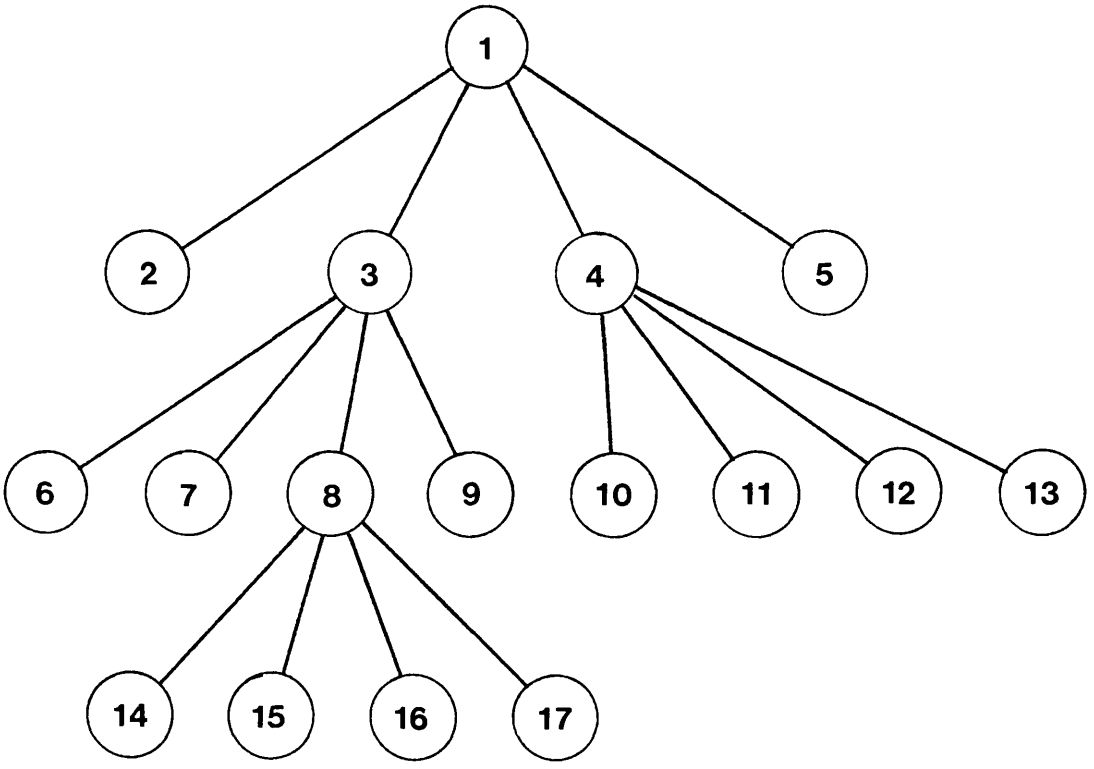
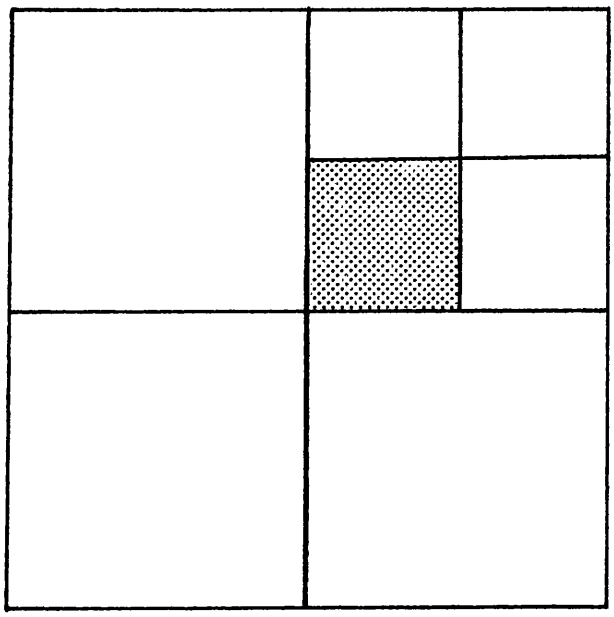


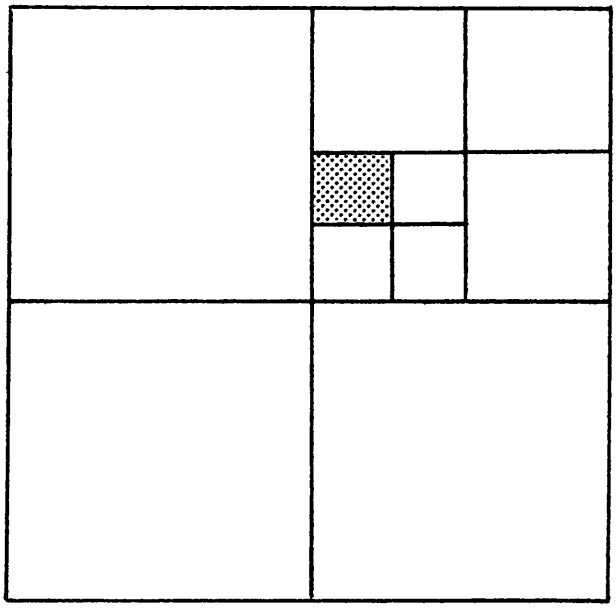

Figure 3.5 The order of nodes in a breadth first traversal

### 3.2.4 Leafcode

In the previously described schemes the size and position of any leaf quad is made unambiguous by context. Leafcodes provide a linear data structure in which only leaf nodes are described and therefore they require some spatial information to be tagged to the colour code of each leaf. Most commonly [38] a field is allocated describing the route taken through the tree to reach the quad. This leaf coordinate can be given by a string of quarternary codes each requiring two bits. For example, by allocating the codes 0,1,2,3 to the quadrants NW, NE, SW and SE respectively the shaded quad in figure 3.6a is given the coordinate 12. Similarly the quad in figure 3.6b is labelled 120. This immediately raises the problem of size ambiguity if the labels are stored in a fixed width field. Gargantini [40] employs an extra code X to indicate no further subdivision, so the quads in figure 3.6 would be labelled 12X and 120. Unfortunately this necessitates a three bit field instead of the previous two for each except the most significant coordinate digit. It is generally more efficient to dedicate a separate size field. For example, a picture resolved to 256 x 256 pixels requires an eight digit coordinate to specify a particular pixel. In Gargantini's labelling scheme the coordinate field width is therefore $7x3+2 = 23$ bits. Alternatively, since there are eight possible quad sizes the size can be denoted in a three bit field and the total requirement is then $8x2+3 = 19$ bits.



a) leaf 12 or 12X shaded



b) leaf 120 shaded

Figure 3.6 Labelling quadtree leaves

To summarise then, a leafcode might typically be structured as a list of the form

| | position code | size code | colour code |
|---|---|---|---|
| quad 0: | ..... | ..... | ..... |
| quad 1: | ..... | ..... | ..... |
| etc | | | |

and ordered by position code.

The overheads imposed by the storage of position and size codes make this structure look fairly unattractive. However, it does offer some benefits. Firstly, if the image in question can conveniently be partitioned into subject and background then only subject quads need be stored; the background can be inferred. Secondly, the structure allows rapid random access to any quad whose screen (x,y) coordinates are given (Appendix 1). Access can be made either by binary search or by setting up a hash table using, in both cases, position code as a key.

Oliver and Wiseman [41] claim that leafcode provides a superior data structure for performing translate, scale and rotate operations on images represented as quadtrees. Their algorithms employ squarecode: a generalised leafcode which permits a picture to be represented by any tesselation of different sized squares. In order to maintain complete flexibility they propose a system which stores treecode and converts to and from squarecode via leafcode whenever this promises greater efficiency.

It is worthwhile to note that a leafcode description which includes both position and size codes contains redundant information which may be exploited by implicit ordering. There is

clearly some scope for compression since, given a full list of position codes the associated size codes can be deduced. The author has employed a leafcode which dispenses altogether with position code retaining only colour and size information. This leafcode, named the Display Ordered List, is particularly suited to the task of display generation and is described in the next chapter.

## 3.3 Quadtrees as display structures

The quadtree as a structure for display purposes has received little attention though Oliver and Wiseman [38] remark upon its potential for display at varying resolutions, with particular reference to animation.

Samet has clearly considered the question of display and presents an algorithm for converting from quadtree to raster form [30]. His objective, however, (though not specifically stated) is undoubtedly to generate data for a frame store rather than to perform real-time scan conversion. Kawaguchi and Endo [34] give details of a raster to quadtree encoding algorithm which may be inverted to perform decoding.

The quadtree method appears to satisfy three of the principal criteria previously stipulated (2.6) for a computer graphics coding scheme:

(i) it promises to offer data compression comparable with other exact coding schemes,

(ii) it is applicable in principle to any picture type and

(iii) it provides a potentially useful picture representation.

Two issues remain to be settled. The first is to determine to what extent quadtree code (in one form or another) is suitable as a source for display generation. The second is to investigate the extent to which a quadtree based display system can be used interactively.

# Chapter 4 A SCHEME FOR REAL-TIME DISPLAY

Having discussed a variety of data structures which have proved useful in representing pictures stored as quadtrees it now becomes appropriate to focus attention upon the practical problems associated with real-time display. This chapter describes the process which culminated in the current design and thereby arrives at a preliminary specification for the hardware which will be described in the next chapter.

## 4.1 Objectives

The design started with the objective of finding a solution which would satisfy the following criteria.

a) The system should be capable of generating, in real-time, images containing a wide range of colour from compressed picture code. The use of a standard data monitor as the output device minimises cost but imposes a limit on the image resolution which was chosen to be 512 x 512 pixels, being a convenient format and close to the practical limit.

b) It should be simple and cost effective. Some of the advantages offered by the use of compressed code would clearly be nullified by the need for elaborate and extensive hardware. In particular, the use of high performance integrated circuits should be avoided where possible to minimise the cost.

c) It should be robust. Several previous schemes using compressed code have depended for their successful operation on limiting picture complexity (2.5.5). A strategy which is defeated by

critical picture types was considered to be unsatisfactory. This does not imply, of course, that all picture types can be expected to benefit from data compression; simply that the compressed form, however inefficient, should always be susceptible to real-time decompression.

d) As a research tool, the system should retain as much flexibility as possible to allow the evaluation of different coding strategies.

## 4.2 Initial Concepts

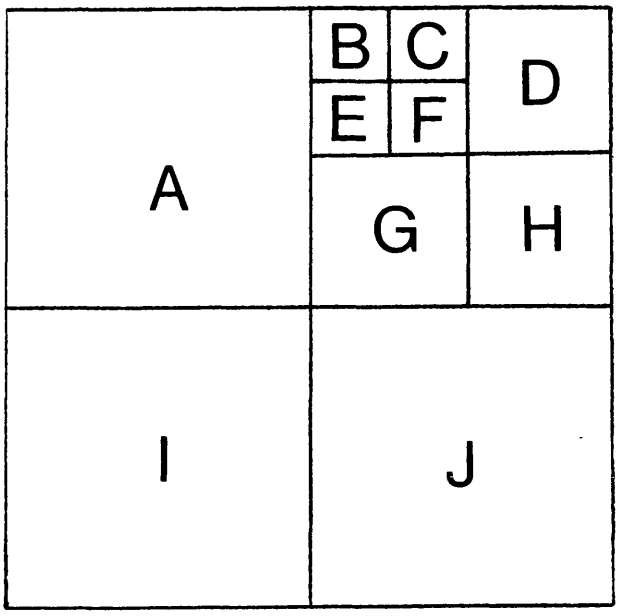
Adopting a bottom-up approach, the design commenced by considering the demand for picture data as the raster scan proceeds in the conventional manner from screen top left to bottom right. Figure 4.1 is provided as a simple example to illustrate the argument.

Starting to scan the first line of a new raster field, the display requires data pertaining to quads A,B,C and D; in particular the colour of and run length across each area must be supplied. Proceeding down the screen successive scan lines require identical data and, rather than fetch this repeatedly, it appears sensible to buffer the data locally. This necessitates the local storage of additional information which will control the entry of data associated with quads E and F when the scan reaches the appropriate line. Eventually, quads E, F and D will be discarded and replaced by G and H.

The preceding analysis suggests two features which will simplify operation of a practical system.

a) Given a small cache memory the picture quads need be fetched

a)




b)

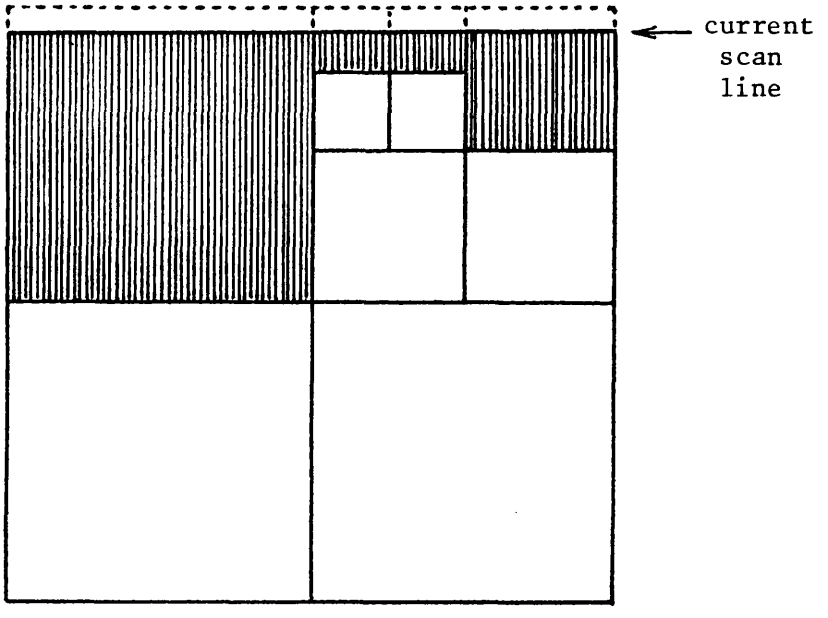



Figure 4.1 a) Display order of quads
b) Picture profile

once only from main storage but in an order determined by the raster representation of the picture. This will be referred to in future as DISPLAY ORDER and corresponds to the order in which the picture quads are first encountered in a top left to bottom right traversal.

b) The size of cache memory required must be sufficient to buffer data for a complete raster line throughout the field. In order to satisfy every eventuality and retain simplicity this LINE BUFFER should provide 512 locations corresponding to the horizontal pixel resolution and should store the COLOUR CODE of each pixel in the current raster line. Furthermore, to control the replacement of colour data in the buffer as the scan proceeds it is necessary to keep a record of the current picture "profile" (Fig.4.1b). This is most easily done by associating with each buffer entry a value denoting the number of lines for which the stored colour remains valid. This value is designated HEIGHT CODE.

A feasible scheme for hardware implementation thus emerges which would operate as follows. During a raster line each buffer location is examined sequentially at pixel rate and the colour code is output. Simultaneously the height code is tested and, if found to be non-zero, is decremented in preparation for the next raster line. However, detection of a zero height code causes the insertion into the line buffer of new colour and height codes associated with the next quad in the display order. This is termed the INPUT QUAD. Consider, for example, the lifespan in the line buffer of an 8 x 8 quad which appears on four raster lines of each interlaced field.

Its colour code is written to eight consecutive buffer locations together with an initial height code of 3. At the fourth pass through the line buffer each of the eight locations contains a zero height code so each must be overwritten with new data. It is thus apparent that, in general, the input quad is used to update several consecutive line buffer locations as they become invalid so a suitable control mechanism is required to permit exactly the correct number of insertions. This can be achieved by associating with each input quad a WIDTH CODE which is decremented each time the quad is used to fill a vacancy in the line buffer. Whenever the width code becomes zero the input quad is replaced by its successor in the display order. Notice that, although they operate in a similar fashion, the height and width codes are quite distinct. The height code is stored in the line buffer and controls requests for new data. The width code is held outside the buffer and controls updating of the input quad by counting line buffer requests.

The height and width codes introduced in the foregoing discussion are both derived from a more general and efficient code for representing quad size. This will be described after examining some of the other factors which have an influence on the design strategy.

### 4.3 Timing Considerations.

Preliminary analysis of pictures provided by colleagues in the School of Engineering at this University revealed that pictures of reasonable complexity were likely to be composed of more than ten thousand quads (Chapter 7). Given a field period of 20ms the

average time allowed to access each quad in the display order is therefore less than 2μs with transient demand for data more than an order of magnitude greater. None of the data structures so far described (Chapter 3) is easily traversed in display order so their direct use would require both:

a) ultra fast processing far beyond the power of any current microprocessor and

b) considerable buffering to even out the demand for quad data by the display.

Given these complications and the previously stated requirement of relatively straightforward hardware it became clear that the quad data should be sorted into display order in advance of real-time display. The decision was therefore taken to adopt the DISPLAY ORDERED LIST as the central data structure, permitting:

a) simple interfacing to the code decompression system and

b) the investigation of compilation from other data structures

The structure of the display ordered list is examined later (4.5).

## 4.4 The Effect of Display Interlace.

Raster interlace is a technique commonly employed in low cost display monitors to realise twice the resolution for a given line frequency. The current display apparatus incorporates this type of monitor and, as a consequence, some of the discussion in the previous paragraphs relating to display order needs slight qualification. The complication arises from the fact that an individual pixel appears on alternate raster fields (fig. 4.2) and therefore pixel-sized quads must be treated as a special case when

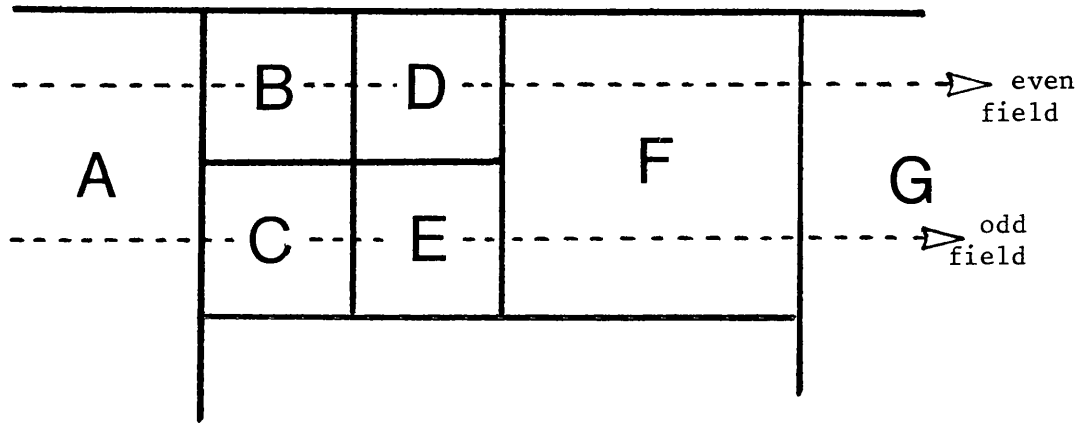



Figure 4.2 Display order of pixel-sized quads

defining display order. Bearing in mind that the screen is refreshed on every field rather than on every frame two solutions are suggested.

a) The concept of display order defined previously is extended by defining separate even field and odd field display orders. Since the possibility of real-time ordering from another structure has been currently rejected the implication is that two separate display ordered lists are needed for the display of a picture frame. This would clearly be a very inefficient solution since all the quads larger than pixel size would be included in both lists. For example, in figure 4.2 :

the even field order would be ABDFG and

the odd field order would be ACEFG

A simple calculation based on the statistics presented in Chapter 7 reveals that since roughly 60% of the quads composing a picture are pixel sized and because these divide equally between even and odd raster fields the ratio

number of quads per field : number of quads per frame = 70:100

from which it follows that

length of dual field list : length of single frame list = 140:100.

The use of separate display lists therefore requires an extra 40% storage and was considered an unsuitable proposal.

b) Referring again to figure 4.2 it is observed that the order ABCDEFG includes as subsets the field orders ABDFG and ACEFG. It

would therefore be possible to use a single frame display ordered list defined in this fashion provided the pixels associated with different fields could be identified and "filtered" at display time. This solution was adopted and the hardware implementation is described in the next chapter.

## 4.5 The Display Ordered List.

Having defined the concept of display order for an interlaced raster display it is possible to make detailed proposals for the structure of display ordered lists. Each record in the list must include information on quad colour and, since it is not implicit in this structure, quad size.

The coding of quad size is made particularly efficient by the fact that all quads have dimensions which are a power of two. Starting with pixel size (2^0 x 2^0), which requires two separate codes to differentiate even and odd field, and continuing up to full screen (2^9 x 2^9) results in eleven different codes. If quads greater than 64 x 64 are disallowed in the scheme the number of codes is limited to eight and the size information may be contained in a three bit field. The upper limit placed on quad size imposes negligible loss of efficiency in coding since the occurrence of larger quads is rare and their replacement by several smaller ones adds very little to the length of the display list. The designation of SIZE CODE is given in table 4.1 together with the translation for each quad size into the height and width codes required by the line buffer.

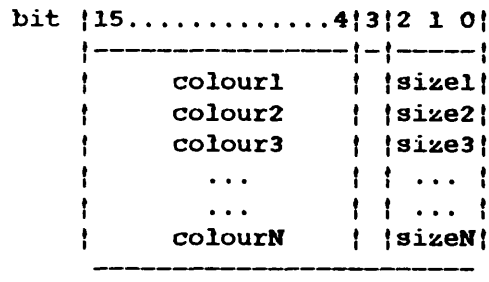
| Quad Dimensions | Size | Height | Width |
|---|---|---|---|
| Even field pixel | 0 | 0 | 0 |
| Odd field pixel | 1 | 0 | 0 |
| 2 x 2 pixels | 2 | 0 | 1 |
| 4 x 4 pixels | 3 | 1 | 3 |
| 8 x 8 pixels | 4 | 3 | 7 |
| 16 x 16 pixels | 5 | 7 | 15 |
| 32 x 32 pixels | 6 | 15 | 31 |
| 64 x 64 pixels | 7 | 31 | 63 |

Table 4.1 Quad codes

Provision for colour in the display is determined by the number of bits used in the colour code; for example, a range of 256 colours on-screen requires a code of eight bits. The eventual choice of colour resolution must of course depend upon the intended application but in this system the effect of exploiting area coherence should also be considered. Is it, for example, worthwhile to implement a colour range greater than the anticipated maximum number of quads? The question of colour is examined in some detail in Appendix 2 but in relation to the present system it was decided to employ a colour code which offered a substantial range within a convenient data field. Having reserved 3 bits for the size code the use of an 8 bit record for the display list would leave a maximum of 5 bits for the colour code. This was considered inadequate so the decision was made to use a 12 bit colour code within a 16 bit record leaving one bit spare for possible future use. Although this may be considered excessive in many applications it leaves scope for experimentation in the representation of colour (Appendix 2). Yet more flexibility is provided by the inclusion of a hardware

colour look-up table which maps the 12 bit colour code to a 24 bit code consisting of three 8 bit values. These values determine the intensity of each of the primary colours (red,green,blue) mixed to produce the colour associated with a particular 12 bit code. Thus, the present design is capable of displaying a range of 4096 colours selected from a total of more than 16 million (2^24).

The display ordered list therefore consists of a sequence of 16 bit leafcode records structured as:




Bit 3 is currently unused.

It has already been noted that the length of the display ordered list is likely to exceed ten thousand records. The question arises what provision should be made in the hardware for storage of the list. This may be answered with reference to the degree of compression considered acceptable in the system. A frame store with the same resolution as the proposed display requires 512 x 512 ie. 256K words of storage where the word length determines colour resolution. Provision in the present system for a 32K record display list therefore caters for any picture in which the ratio of pixels:quads is better than 8:1.

## Chapter 5 THE SYSTEM HARDWARE

While hinting at some wider aspects, the design considerations discussed in the preceding chapter effectively concern only the part of the hardware dedicated to real-time display from the compressed picture code. The rest of the hardware must, by virtue of the research nature of the project, be specified to enable maximum flexibility in the investigation of picture coding and manipulation.

In connection with a separate project the author had already designed hardware for a graphics system employing a full bit-mapped framestore. This Z80 based system, supported in the department by suitable software and expertise, provided an excellent test-bed for a preliminary implementation of the design strategy outlined in Chapter 4. Having stripped the system of redundant features, design and construction of the picture buffer and decoder commenced. A block diagram of the initial prototype is shown in figure 5.1. The picture buffer, whose function is to store the display ordered list of quads, was limited in this prototype to hold up to 16K quad records each with a 4 bit colour code. Operation of the picture decoder was tested using geometric patterns whose display lists were loaded quad by quad into the picture buffer under the control of the system monitor. Lacking a colour map, the 4 bit colour codes were fed directly to a single digital to analogue converter (DAC) which, in turn, drove a monochrome data monitor producing a sixteen level grey-scale image. With the assurance that the decoding circuitry was performing to specification it became necessary to

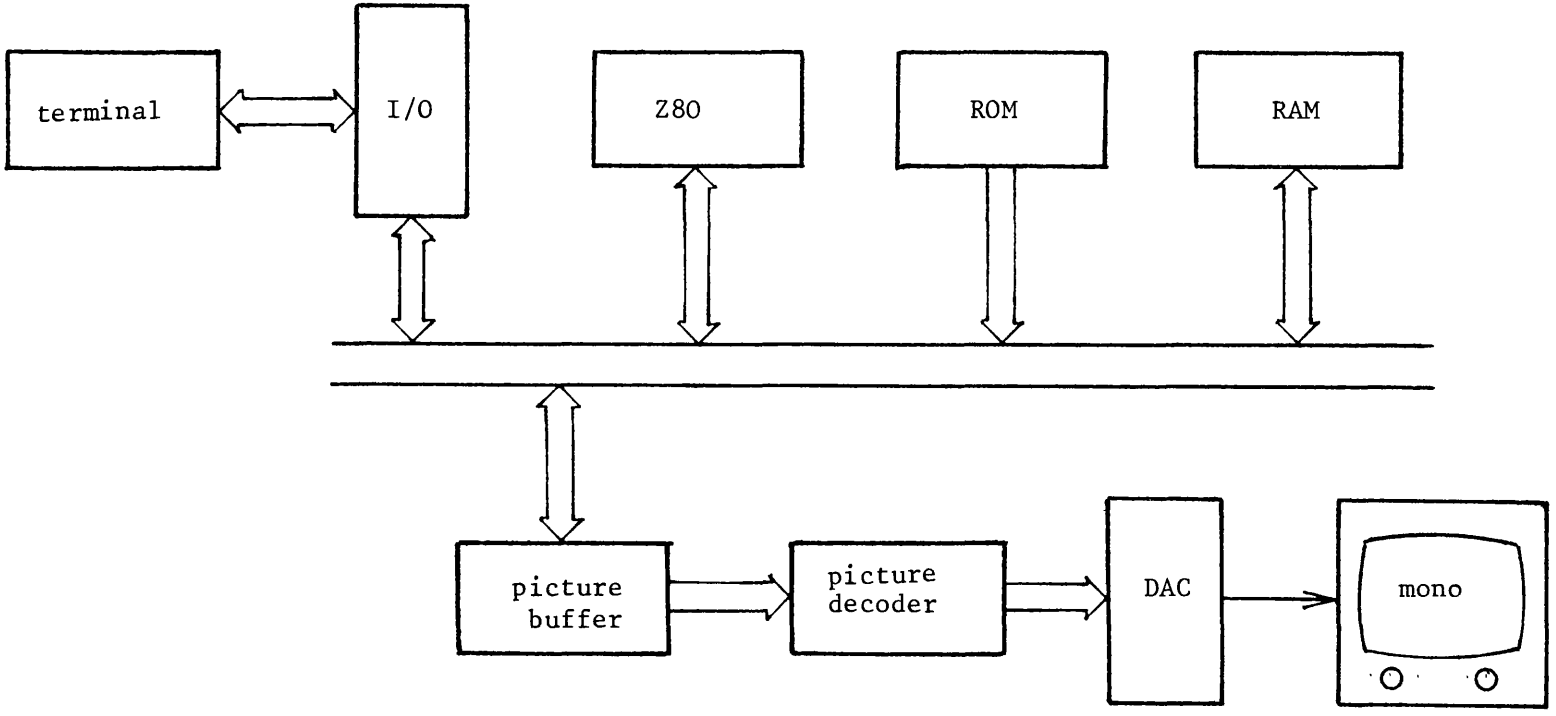



Figure 5.1 The Z80 prototype

reappraise the situation. Two major deficiencies in the prototype system had become apparent, both of them related to the use of the Z80 as CPU.

(i) In order to extend the colour range as proposed, each quad record in the display list would require two bytes of storage with provision for a total of 64 Kbytes. Transfer from an 8 bit to a 16 bit processor would clearly make operations on the display list easier and more rapid.

(ii) The Z80 is constrained to address only 64 Kbytes of memory directly. With the display list alone occupying this amount of space the need for bank switching and a consequent management scheme becomes clear. This too can be avoided by transferring to a different CPU.

At about this time a range of 16 bit processors were becoming widely available and one in particular was finding use in other departmental projects. The choice in the subsequent design of the Motorola 68000 as system CPU was decided by features which make it well suited to the present application viz.

(i) its 16 bit external data bus and

(ii) its extensive (16 Mbyte) addressing capability.

Adoption of the new processor, while leaving the decoder design unaffected, necessitated considerable redesign elsewhere and reconstruction provided the opportunity to rationalise parts of the circuitry. The addition of colour facilities and improved input/output interfacing completed the current hardware development whose present configuration is shown in figure 5.2. The remainder

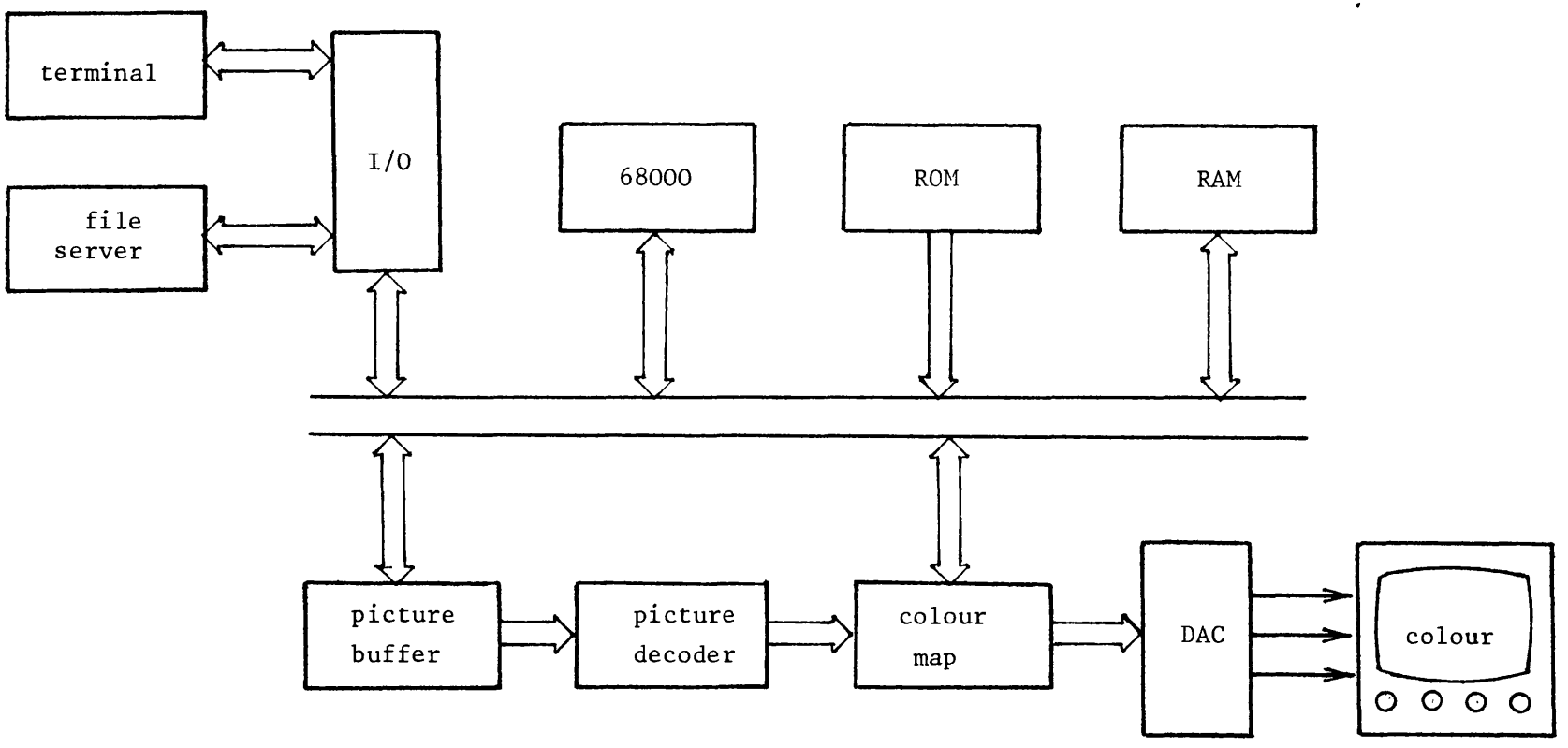



Figure 5.2 The present Picture System hardware

of this chapter is devoted to an examination of the present system hardware. In general, the text is limited to a description in terms of functional blocks. However, it is considered appropriate to include full details of the picture decoder hardware since these are necessary for a complete description of its operation.

## 5.1 The central processing unit (CPU)

Apart from the real-time code decompression the CPU is responsible for all data processing within the system including the supervision of input and output of data. The 68000 CPU operates with an 8 MHz clock and supports a large and flexible instruction set enabling rapid and efficient movement of data. The extensive provision of internal registers minimises the use of memory for storing transient variables and thus speeds execution of the various time-critical routines associated with graphics applications.

## 5.2 Read-only memory (ROM)

This is an area of non-volatile memory in which the system control program resides. Provision is made on the CPU board for 48 Kbytes of firmware organised as 24K words using type 2732 EPROMs with an access time of 350ns. A description of the functions currently provided by the control program is given in Chapter 6.

## 5.3 Random access memory (RAM)

This is an area of read/write memory which is used as workspace for the control program and for the local storage of compressed picture data. The system RAM currently provides 256

Kbytes of storage on a single board using type 4164 dynamic memory. Owing to the increased vulnerability of this type of memory to "soft" errors the board is equipped with error detection and correction (EDC) circuitry. In the present application this is particularly beneficial since a single error in the compressed picture code is likely to propagate with disastrous effect throughout the picture. The use of EDC unfortunately increases the memory access time to 500ns but this has not proved a serious drawback. The present memory size allows storage of (typically) ten pictures but provision is made in the racking to accommodate a second board which will double the capacity.

## 5.4 Input/Output interface (I/O)

This allows communication between the control program and external devices. One of the most important benefits to be gained from the use of compressed picture code is efficient transmission via a low bandwidth line. Accordingly, the system is provided with an RS232 serial port for connection to a host computer or a network. In general, the system has been operated in conjunction with a simple Z80 microcomputer which manages picture file storage on mini floppy disks under the CP/M operating system. In addition to the serial line, which operates at speeds up to 19200 baud, a parallel interface with the host provides considerably more rapid communication. A second RS232 serial interface is used for commmunication with a terminal through which system commands are issued. Recently the Z80 computer has fulfilled the dual role of terminal and file server.

## 5.5 Picture buffer

This is an area of RAM used to store the display ordered list of the currently displayed picture. During the raster field period (20ms) the entire display list held in this memory is accessed sequentially by the picture decoder at a rate sufficient to maintain its correct operation. In the worst possible case where the display list contains a succession of pixel size quads this implies access at a rate equal to twice the pixel frequency because, as explained in Chapter 4, only half of the data will be valid for a given field. With a pixel period of about 70ns, access time to the memory is therefore set at 35ns per quad. The design of the picture buffer (fig. 5.3) incorporates a parallel fetch of data for eight quads thereby satisfying the worst case access time of 280ns without the need for costly high speed memory devices. Avoiding the complications associated with dynamic memory the picture buffer uses type 6116 static RAM configured as 4k x 8 x 16 bits providing storage for a picture composed of up to 32K quads. During decoder access a 12 bit address is supplied to read data from a selected block of 8 x 16 bits. Access by the CPU is given priority and in this event the lower 3 bits of a 15 bit address supplied by the processor select one of a block of eight locations. Naturally, any modification of the display ordered list has a disastrous effect on the displayed image so blanking is applied to the video output on these occasions.

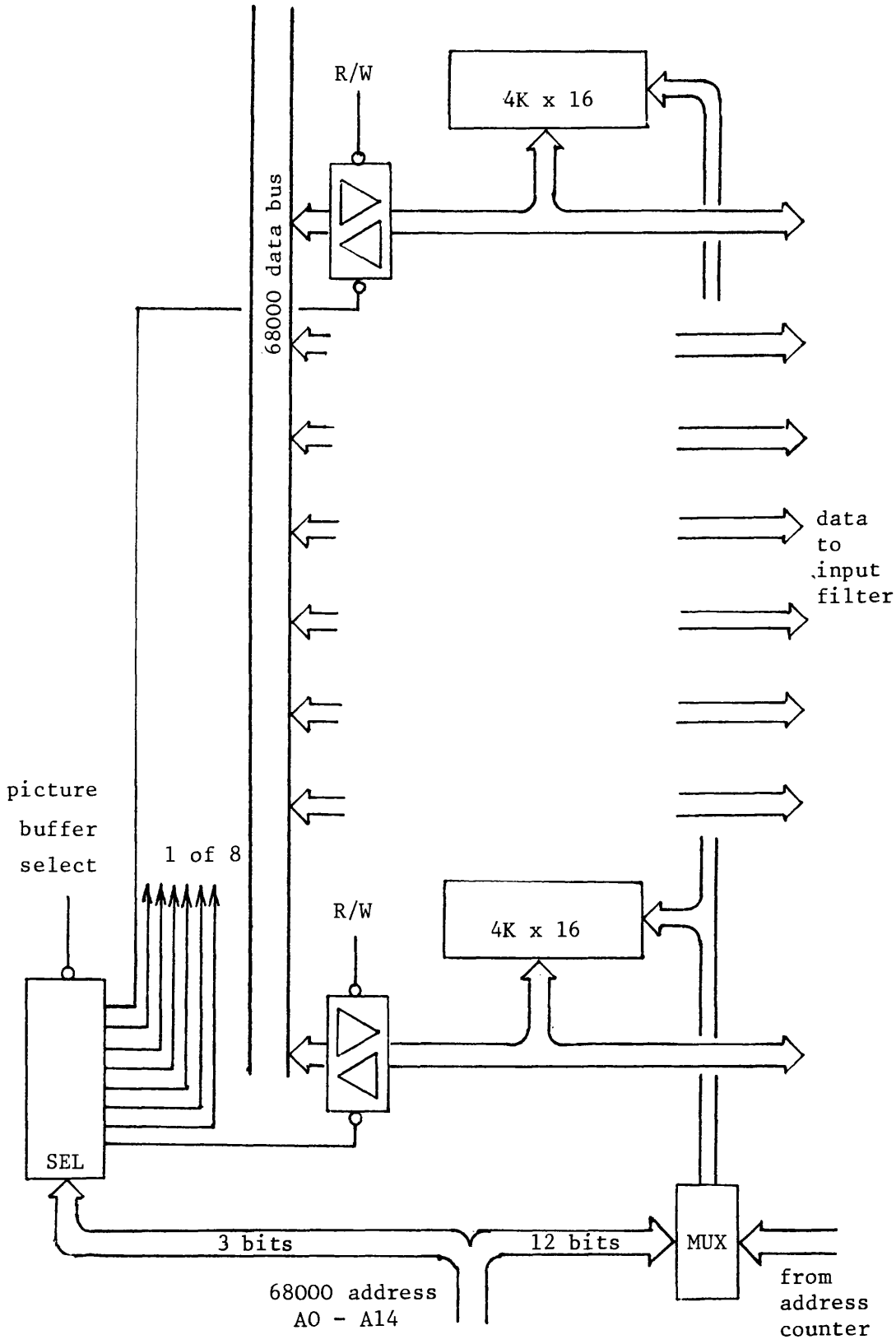



Figure 5.3 Picture buffer

## 5.6 Picture decoder

The function of this block is to generate a real-time sequence of pixel colour codes from the compressed picture code stored in the picture buffer. The individual elements of the decoder shown in figure 5.4 will now be described in detail.

### 5.6.1 Line buffer

(i) Functional outline

The principles underlying the operation of the line buffer have already been presented in Chapter 4. This section continues logically from there by examining the practical implementation of those ideas. Recall that the buffer requires 512 locations in which are stored the colour and height codes for the current raster line. As the line is scanned each buffer address is stable for about 70ns during which time

a) the colour code must be read and output

b) the height code must be read and

(i) if not zero, decremented

or (ii) if equal to zero, initiate the writing of new colour and height codes

The execution of a read/decrement/write cycle in the specified time proved to be outside the capability of any readily available fast memory. Subsequent examination of the benefit of double buffering quickly provided a solution which effectively lengthened the cycle time to a manageable duration. Under this scheme colour and height codes are read from one of the buffers (the R buffer) while the other (the W buffer) is written with codes appropriate to the

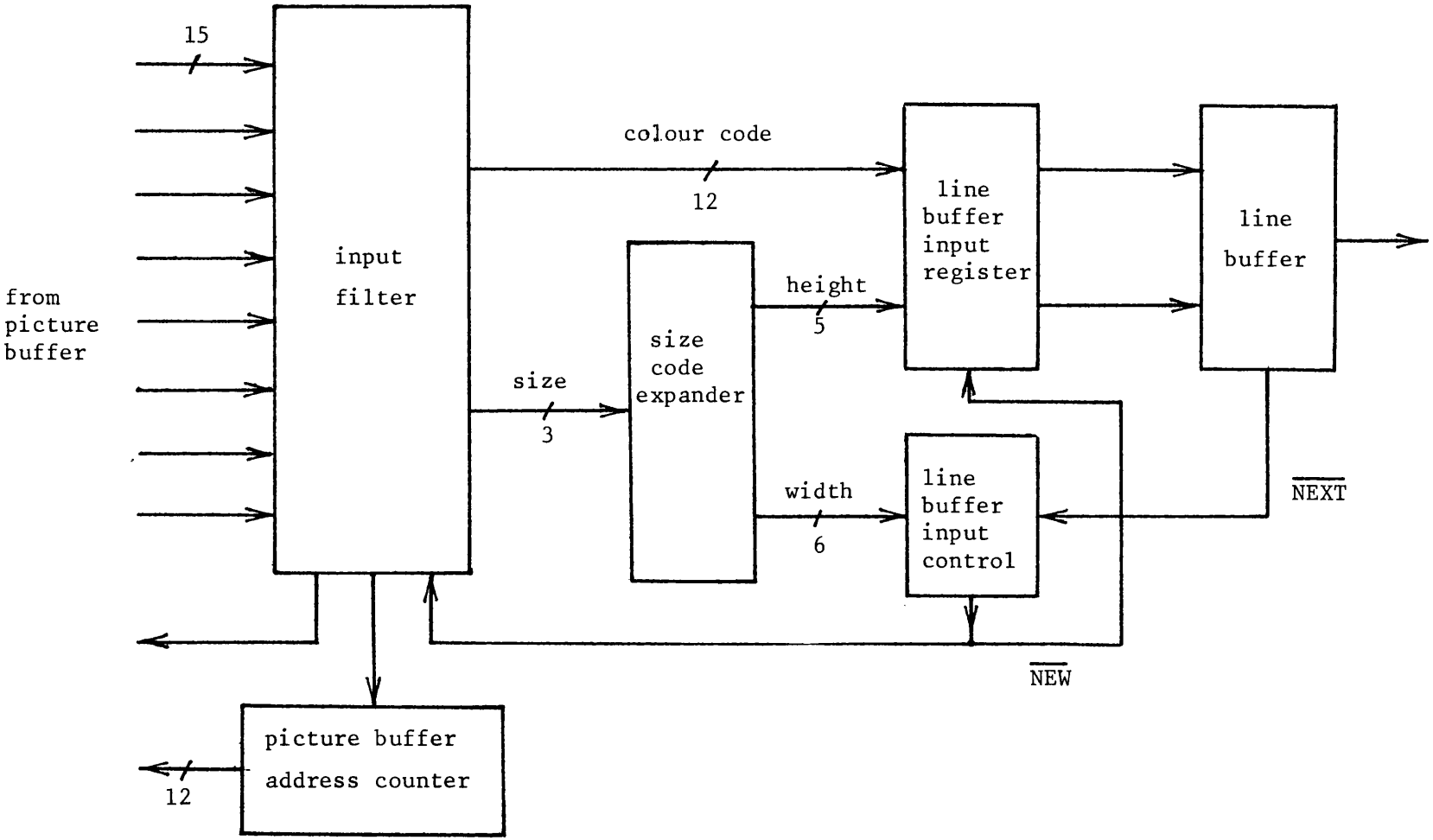



Figure 5.4 Picture decoder

subsequent raster line (fig. 5.5). Height code from the R buffer is passed to a fast decrement circuit whose output is routed through a multiplexer to the W buffer input. The decrement unit is used to control the multiplexer so that new height code is fed to the W buffer whenever a zero height code is detected.

On alternate raster lines the functions of the R buffer and W buffer must clearly be reversed. Figure 5.6 shows how this is achieved using the control signal $\overline{A}/B$ to select which of the buffers A and B is designated the R buffer during the current line. During the vertical blanking period data for the first raster line is written to buffer A. Each display field therefore starts by reading from buffer A and writing to buffer B. Notice that provision is made for copying colour code from buffer A to buffer B but not the reverse. This is because new code is only written to buffer B in the case of size 1 x 1 and size 2 x 2 quads which persist for just one raster line.

(ii) Operational details

Line buffer control (fig. 5.7) is synchronised to the main display clock DCLK running at pixel frequency. XOR gating with signal $\overline{A}/B$ generates two antiphased clocks CLKA and CLKB which are used to latch column addresses (DA0-DA8) providing two sets of addresses for the double line buffer. The advanced address is applied to the R buffer while the delayed address is applied to the W buffer effectively increasing the cycle time to more than 100ns. Gating of DCLK with various condition signals provides a variety of write pulses to different parts of the circuit.

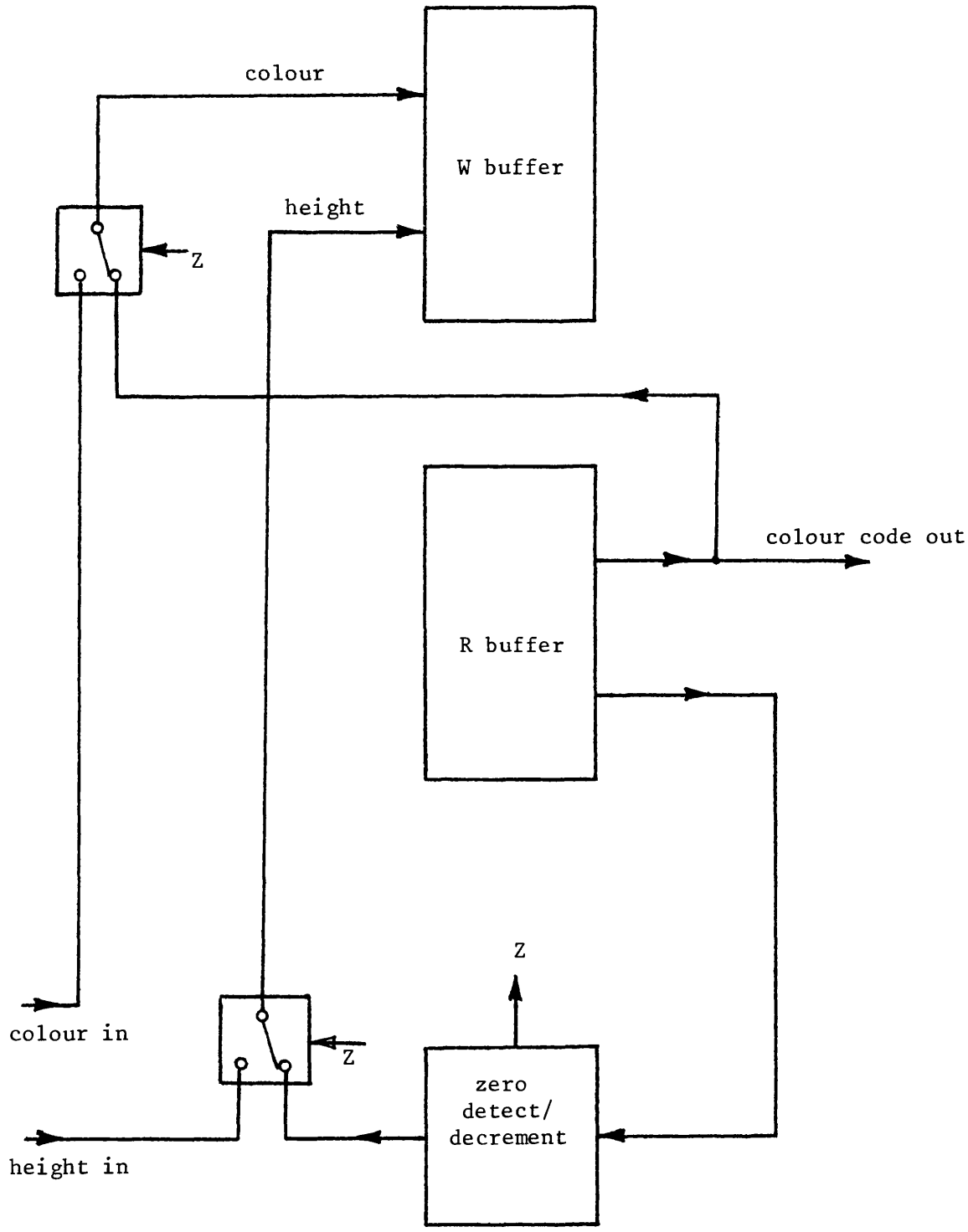



Figure 5.5 Line buffer R/W operation

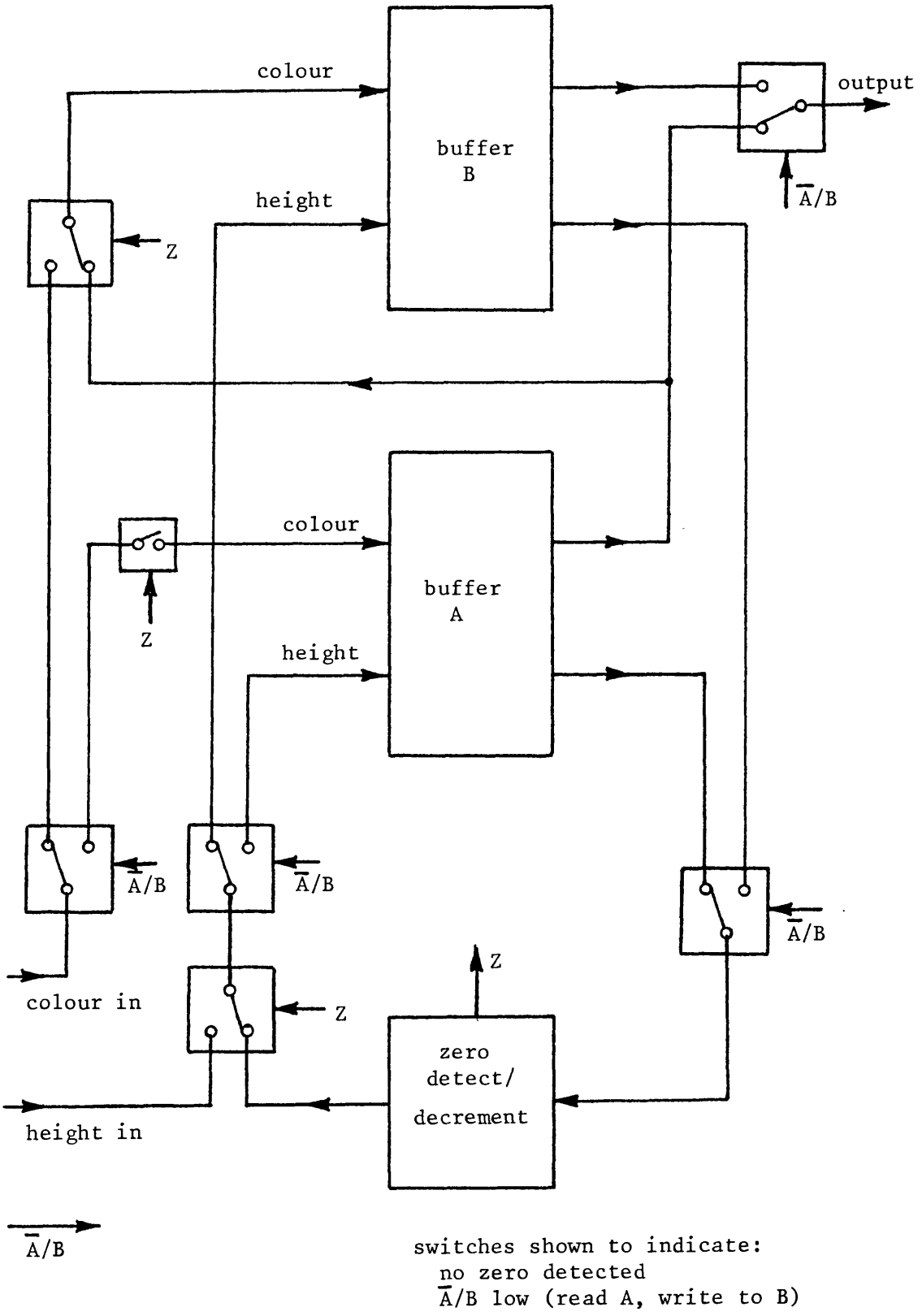



Figure 5.6 Line buffer A/B operation

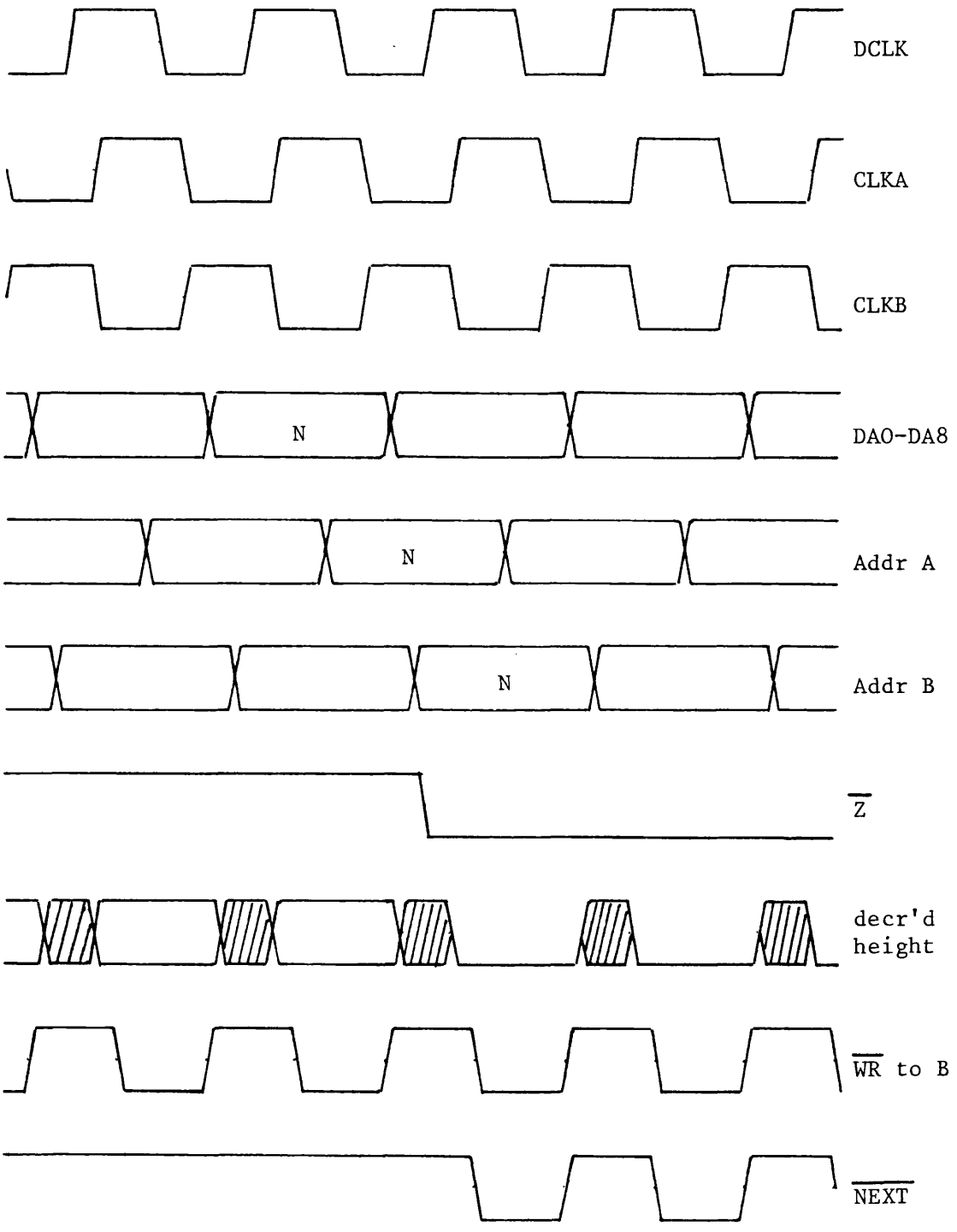



Figure 5.7 Line buffer signal timing

Height code is stored in type 2125 memory with an access time of 20ns. Data selected from buffers A or B is passed to the fast decrementer (fig. 5.8) which asserts signal ZERO if the height code is zero. This signal is latched during the write phase of the cycle providing signal Z which is used to enable height data from either the decrementer or from the input register . A request NEXT is issued by the line buffer each time data is written from the input register.

Colour code is stored in type 2149 memory with an access time of 45ns. During the read phase data is clocked into an output register and, when buffer A is being read, latched in a transfer register for copying to buffer B. In the event of signal Z being asserted colour data is written to the appropriate buffer from the input register.

### 5.6.2 Line buffer input control

The use of width code to control updating of the input quad has been described in Chapter 4. The practical implementation makes use of two 74S169 ICs which form an 8 bit counter wired to operate in count-down mode. The line buffer request signal NEXT is used as a clock to decrement the counter while the input register data remains stable. The counter provides an output which becomes active when the count reaches zero and this is fed back to enable new width code to be loaded on the next clock pulse. The same output is gated with NEXT to generate a signal NEW which loads fresh quad data into the input registers.

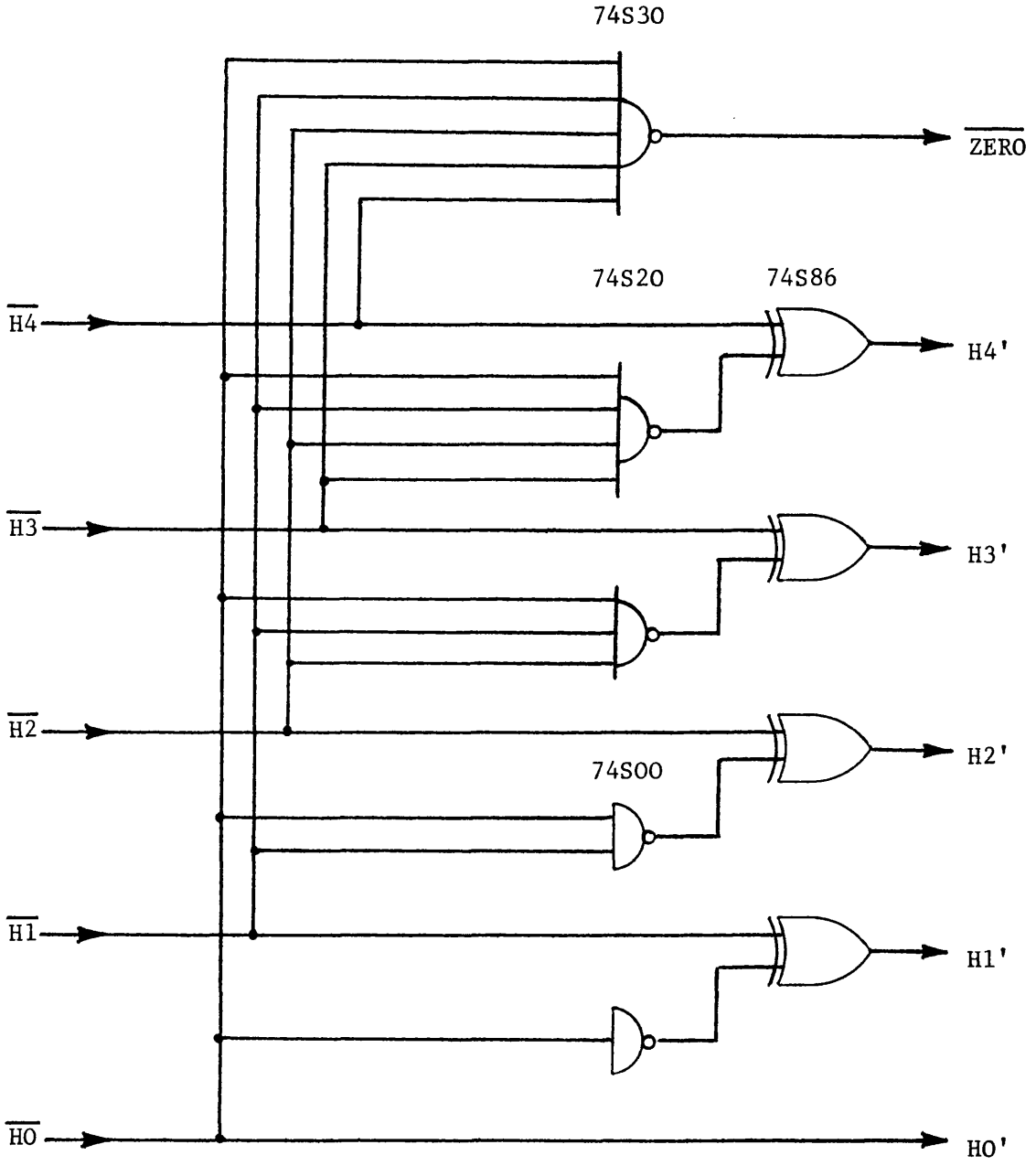



Figure 5.8 Fast decrement circuit

### 5.6.3 Size code expansion

Examination of Table 4.1 which translates from size code to width and height codes reveals that the 5 bit height code is a binary subset of the 6 bit width code, bit 0 being discarded. Figure 5.9 shows how the translation is effected at the expense of just one gate delay through fast TTL.

### 5.6.4 Input filter

(i) Functional outline

The purpose of the input filter is to deliver quad data, accessed in parallel from the picture buffer, to the line buffer input removing any data not relevant to the current field. The parallel to serial conversion therefore requires a selection procedure which recognises and rejects invalid quads. This is implemented in the hardware by an eight to one multiplexer controlled by a simple finite state machine FSM whose output defines the currently active channel (fig. 5.10). Size codes from each of the eight channels are fed to comparators which set a flag V for each channel holding valid data. These flags together with the current machine state determine the next state. With the foreknowledge that invalid quads occur as single items in the display list it is necessary only to examine the validity of the immediately next channel

eg. if channel 4 is currently open then the next channel can be predicted by examining the validity of data on channel 5: if it is valid then channel 5 is opened next; if not then channel 6 is opened next. Expressing the states in binary form:

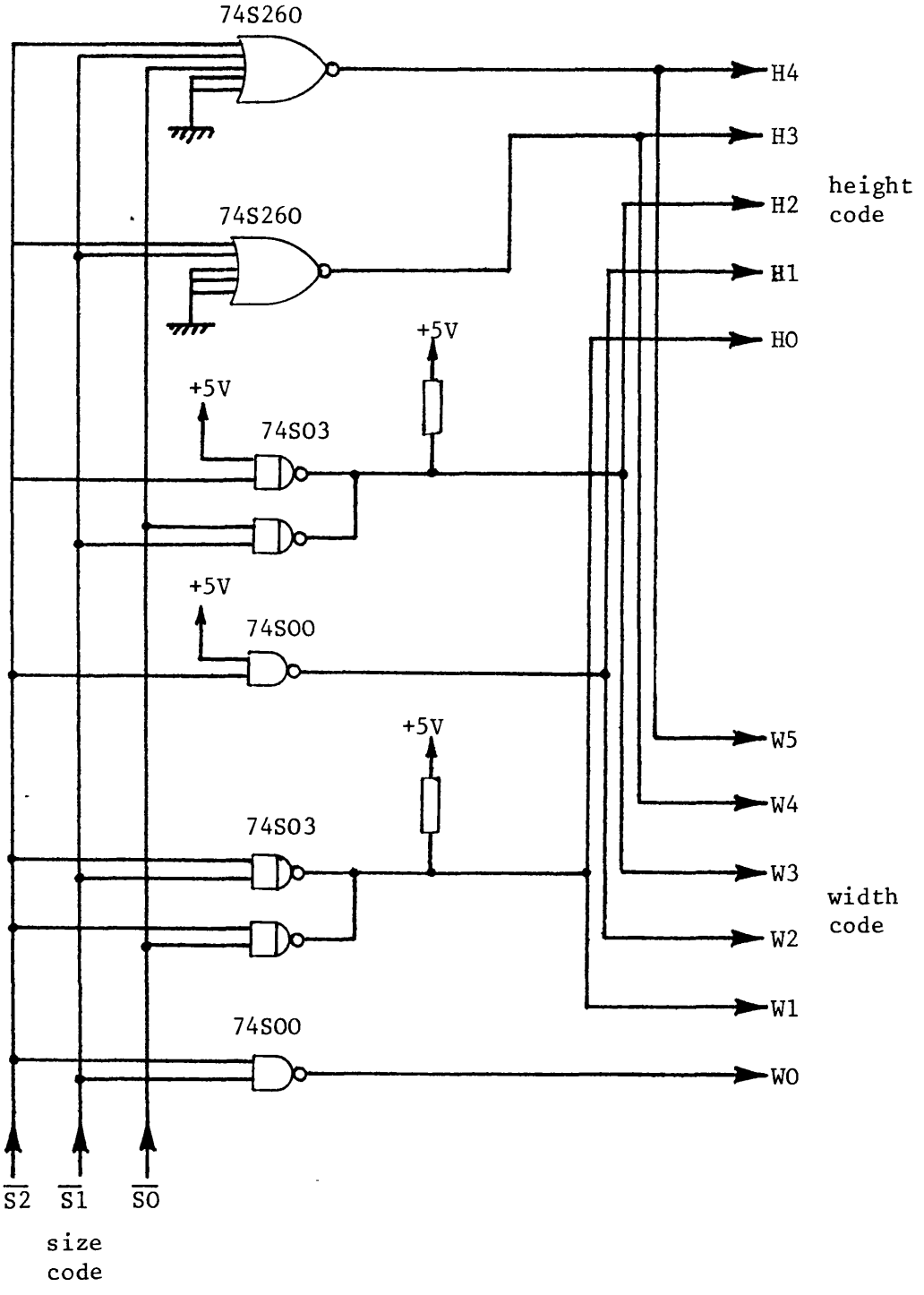



Figure 5.9 Size code expander

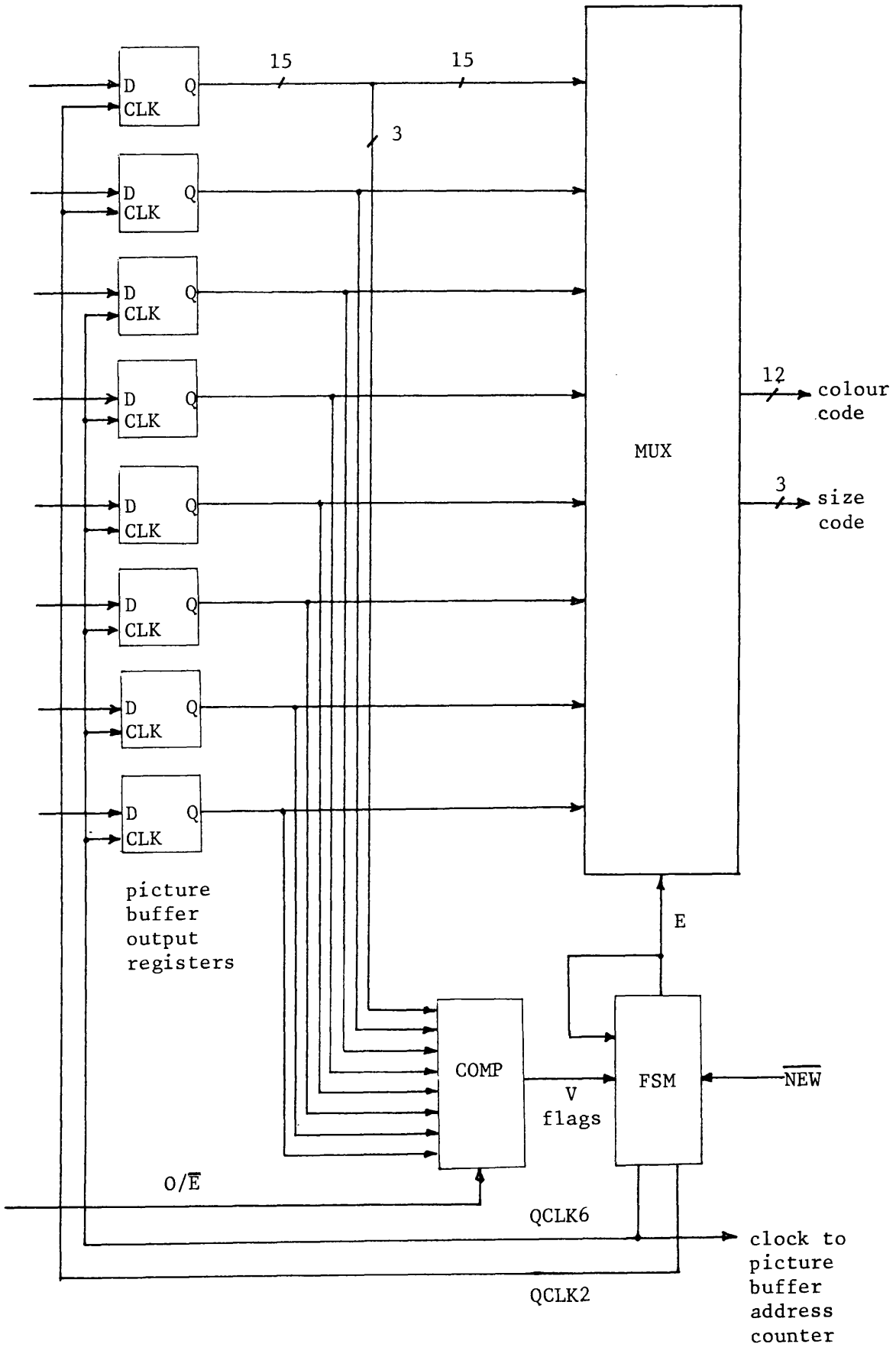



Figure 5.10 Input filter

if S(t) = 100 then if V5 = 1 then S(t+1) = 101

else if V5 = 0 then S(t+1) = 110

so in terms of V5 S(t+1) = 1 $\overline{V5}$ V5

Table 5.1 presents the results of such an analysis for each machine state.

| Current state S(t) | | | Next state S(t+1) | | |
|---|---|---|---|---|---|
| b2 | b1 | b0 | b2 | b1 | b0 |
| 0 | 0 | 0 | 0 | $\overline{V1}$ | V1 |
| 0 | 0 | 1 | 0 | 1 | $\overline{V2}$ |
| 0 | 1 | 0 | $\overline{V3}$ | V3 | V3 |
| 0 | 1 | 1 | 1 | 0 | $\overline{V4}$ |
| 1 | 0 | 0 | 1 | $\overline{V5}$ | V5 |
| 1 | 0 | 1 | 1 | 1 | $\overline{V6}$ |
| 1 | 1 | 0 | V7 | V7 | V7 |
| 1 | 1 | 1 | 0 | 0 | $\overline{V0}$ |

Table 5.1

A consequence of this strategy is that size data on channels 0 and 1 must be prefetched and available while the machine state is 6 or 7.

(ii) Operational details

The valid flag V is generated by simple gating of the size code with the odd/even field signal O/$\overline{E}$. V is set unless bits 1 and 2 of the size code are low (indicating a pixel sized quad) and bit 0 does not match signal O/$\overline{E}$.

Figure 5.11 provides details of the FSM. The current machine state given by the three bit signal E is fed to a multiplexer which selects the appropriate pattern of flags constituting the next state. These form an input to a D-type register which is clocked by the signal NEW thereby opening a new channel after each updating of

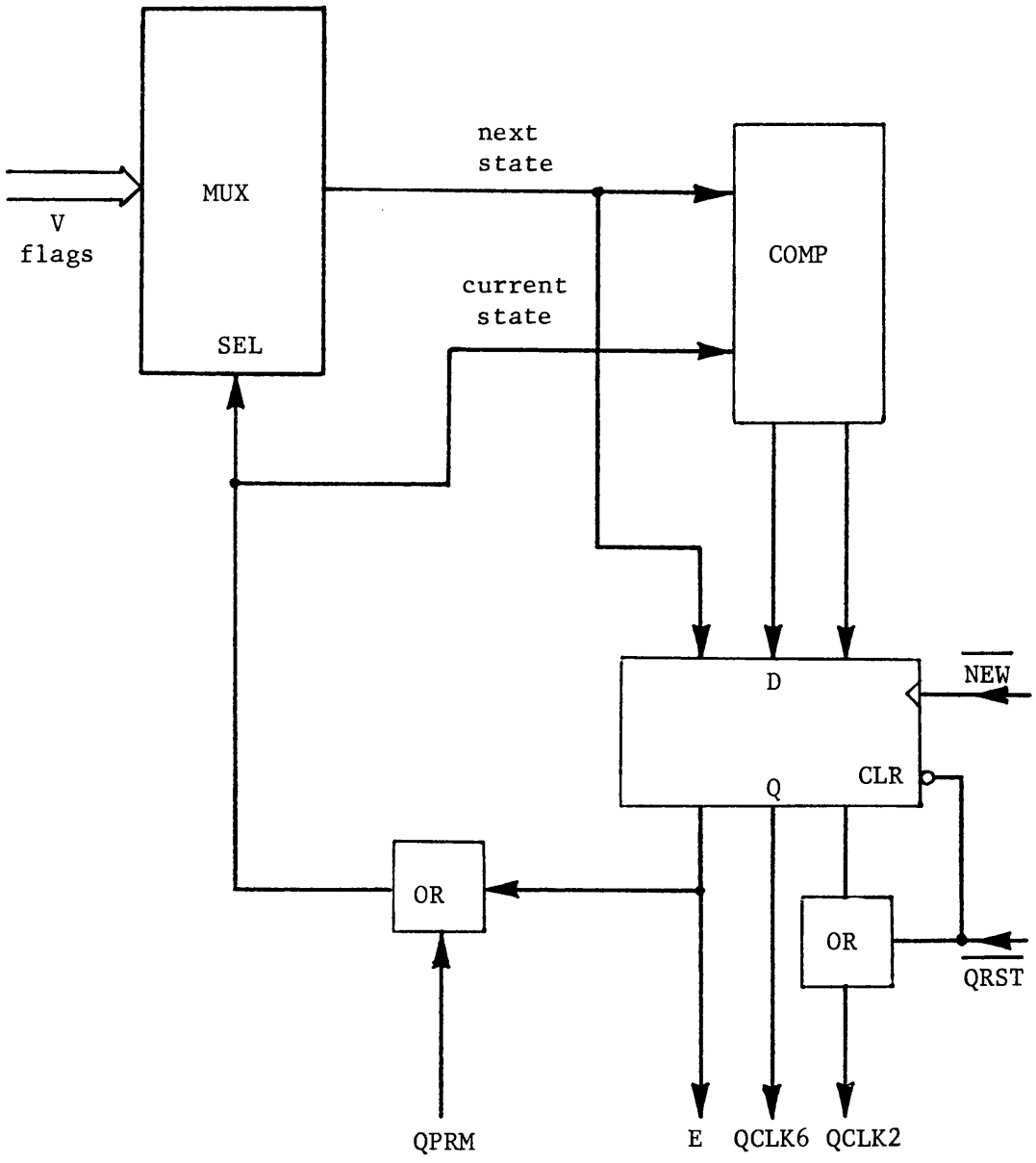



Figure 5.11 Input filter FSM

the line buffer input register.

Further outputs from the FSM are signal QCLK2 which goes high when state 11x (6 or 7) is entered and QCLK6 which goes high when state 00x (0 or 1) is entered from state 11x (6 or 7). QCLK2 is used to implement the prefetch on channels 0 and 1 and QCKL6 latches data for the other channels as well as incrementing the picture buffer block address. Initialisation of the input filter is performed during field blanking by the signals QRST and QPRM whose function is described in section 5.6.5. Multiplexing of the eight data channels into one is controlled by the state signal E.

### 5.6.5 Synchronisation

Signals from this block provide synchronisation between all the activities concerned in the generation and display of the final monitor image. The circuitry is built around a type ZNA134 IC which provides a variety of video synchronisation signals suitable for interlaced raster scan display. Control signals for the decoder are derived from the outputs of two counters. The first counter is clocked at line frequency and divides the field into its display and blanking intervals within which are defined the signals $\overline{A}/B$, LBWR (line buffer write enable), QRST, SETUP, QPRM, and QP2. The timing of these is shown in figure 5.12 and their function, if not described previously, is dealt with shortly. Each line drive pulse initiates the operation of the display dot clock DCLK whose frequency is adjustable to allow alteration of the image width and thus set the display ratio to the required square geometry. A counter driven by DCLK generates the 9 bit column address (DA0-DA8)

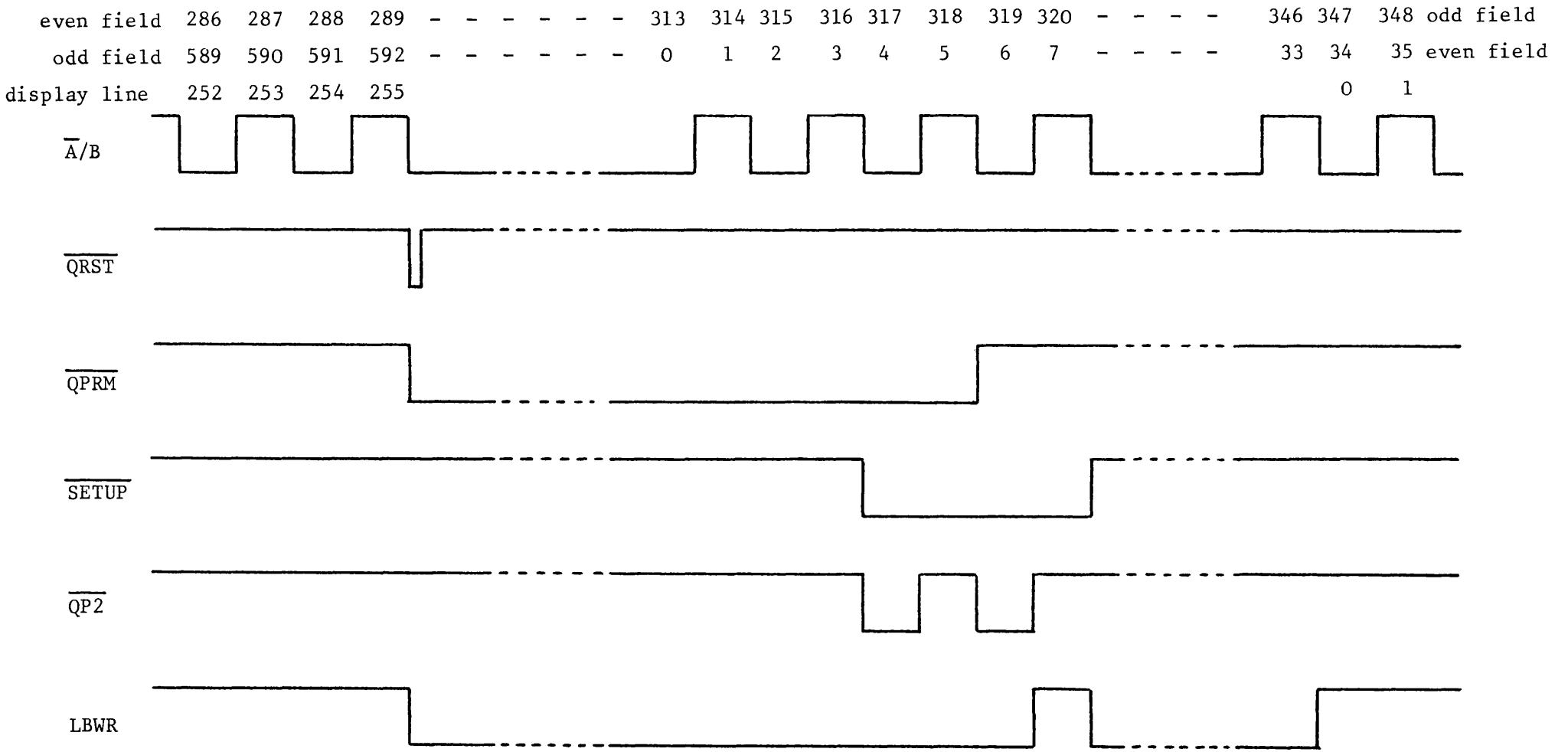



Figure 5.12 Field timing

required by the line buffer. Signals derived from these outputs define the horizontal display and blanking intervals.

Several of the signals generated in this block are concerned with the initialisation of the picture decoder. The input to the line buffer operates in the fashion of a shift register so the first phase of initialisation consists in priming the queue so that the first quad is loaded together with its width code at the line buffer input. This is performed by the signals QRST, QPRM and QP2 as follows. QRST clears the picture buffer address counter, resets the input filter to state 0 and latches data on channels 0 and 1 from the picture buffer. During QPRM the line buffer input counter is forced into a load state and the input filter size code multiplexer is conditioned to output a zero code. The assertion of QPRM also "fakes" a state 7 at the input filter ensuring that the data present on channels 0 and 1 will be inspected to determine the channel opened by the first NEW pulse. QP2 consists of two pulses, the first of which occurs while QPRM is active and loads the line buffer input register with zero width code. It also propagates as NEW thus opening the first channel with valid quad data. The second QP2 pulse then loads this data into the line buffer input register and opens the next input channel. The system is now primed and ready for filling line buffer A with quad data. LBWR is asserted for one line period and together with SETUP, which is active throughout the initialisation period, ensures that each of the 512 locations is written to in preparation for reading on the first display line.

## 5.7 Colour map

This acts as a real-time look-up table which maps colour codes to their appropriate red, green and blue video signal amplitudes. The input consists of a stream of 12 bit colour codes issued by the picture decoder at pixel rate. This provides an address into an area of fast RAM configured as 4K x 3 x 8 bits and employing type 2147 memory with an access time of 45ns. The output consists of three groups of 8 bit values representing the red, green and blue component intensities.

Access by the CPU is provided for the initialisation and subsequent modification of values within the table.

## 5.8 Video interface

This unit converts TTL digital signals to the analogue signals required by the colour data monitor. The 8 bit value associated with each primary colour is input to a digital to analogue converter (DAC) type TRW-1016 which directly drives 75 ohm cable connected to the monitor (R,G or B) input. Negative mixed synchronisation pulses generated by the synchronisation block (5.6.7) are buffered out to a separate monitor input.

# Chapter 6 THE PICTURE SYSTEM

This chapter deals with operational aspects of the Picture System in its current implementation. Following a definition of the data structures employed the firmware is analysed by examining the processing initiated by system commands issued from the keyboard.

## 6.1 An overview

Throughout this thesis the advantages offered by the use of compressed picture code have been emphasised. The system has therefore been organised to demonstrate and quantify the benefits which result from the implementation of quadtree coding. Adequate provision is made for the local storage of several pictures (typically ten) in a variety of formats. All requests to display a picture derive data locally and therefore initiate downloading from the host whenever the local search fails. Experimentation with different data structures is aimed at increasing the flexibility of the system as regards picture format and image manipulation.

## 6.2 Local data structures

### 6.2.1 Display Ordered List

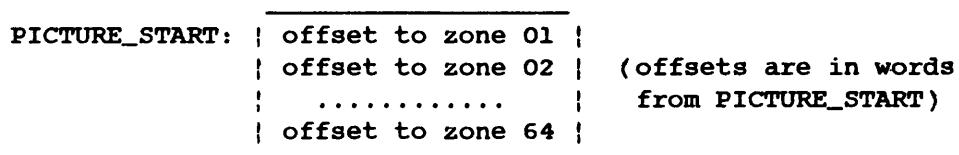
The concept of the Display Ordered List (DOL) has been introduced in Chapter 4 and its structure described in 4.5. The DOL is the fundamental data structure used as input to the picture decoder and is therefore the only valid picture buffer format. The same structure also provides the most economical means of local storage since it contains just one 16 bit record per picture quad and uses no pointers. Furthermore, pictures stored as DOLs are most

readily put on-screen since all that is required is a block movement of data to the picture buffer. The important disadvantage of the DOL is that it obscures the picture structure inherent in the original quadtree and therefore offers little potential for picture processing or manipulation.

### 6.2.2 Zoned Display Ordered List

The strategy of picture zoning adds a degree of flexibility to the DOL. It consists in dividing the screen area into zones each with 64 x 64 pixels thereby creating a display matrix of 8 x 8 zones. While each zone remains an immutable entity the picture as a whole can be manipulated by shuffling zones into an appropriate order. Therefore, it becomes possible to store a large picture composed of m x n zones and roam around it by selecting any subset of 8 x 8 zones for display. The application of zoning frees the quadtree technique from the limitation of representing only square pictures by allowing m x n branches from the tree root node (fig. 6.1).

The zoned DOL for a 512 x 512 pixel picture contains sixty four separate DOLs with added features to facilitate their merging into a single full-frame DOL for display. The zoned DOL is headed by a list of 64 offset pointers to the individual DOLs



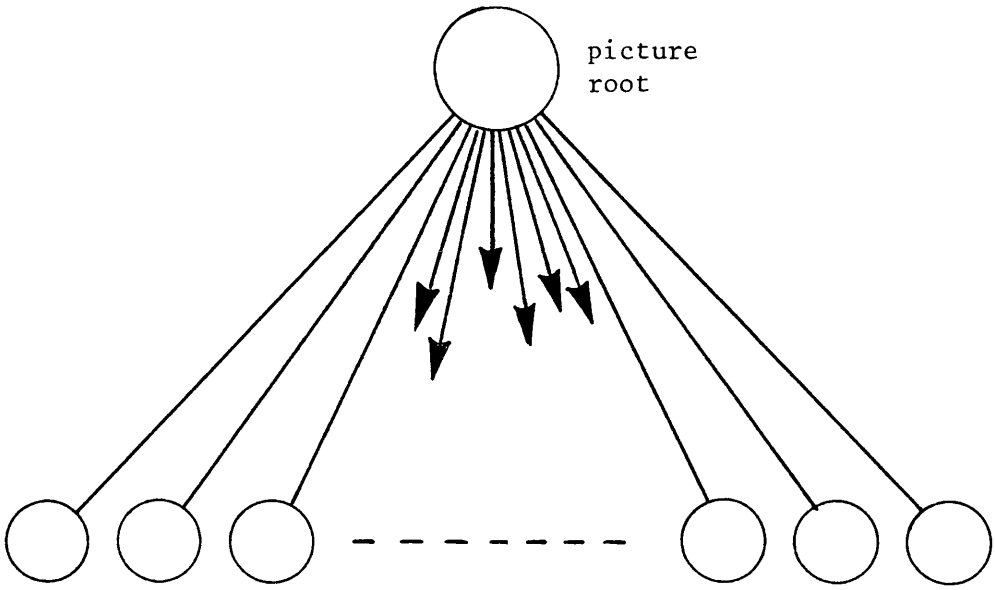
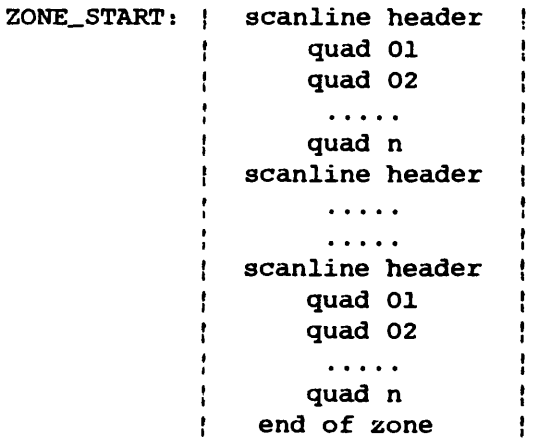



Figure 6.1 Picture zoning

and the DOL for each zone is thus:




where the individual entries are:

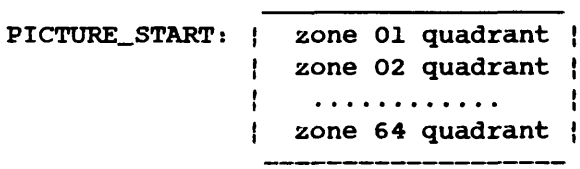
quad word: ccccccccccc0sss — ccccccccccc = colour code; sss = size code

scanline header: nnnnnnnkkkkk1000 — nnnnnnn = (number of quads introduced on this scanline)-1; kkkkk = scanline key

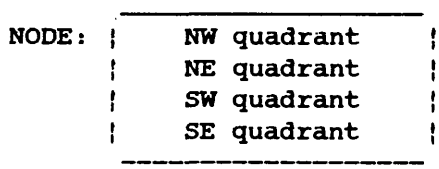
end of zone: 0000000000001111

### 6.2.3 Zoned Quadtree

In this structure the zoning principle is built on to a linked list structure. The structure is entered through a list of sixty-four quadrant codewords which act as zone descriptors. These quadrant codes may either give the zone colour (if it is uniform) or provide an offset to a tree node data cell.



Each tree node data cell is composed of four quadrant codewords thus:




where the quadrant words are in one of two forms;

either colour code: 0000ccccccccccc

or offset to another cell: 1xxxxxxxxxxxxxx

All offsets are in words from PICTURE_START.

The zoned quadtree provides all the advantages mentioned previously in connection with zoned DOLs and retains a structure which permits more complex picture manipulation such as zooming. The penalty paid for this additional flexibility is a lengthier compilation of the frame DOL at display time.

## 6.3 Host picture files

The capture/generation of image data has not been a part of this project. All the pictures which have been used were generated by a solid modelling system under development in the School of Engineering at the University of Bath [35,36] and employ 8 bit colour code. These were made available as full pixel-mapped files on a PDP11/23 minicomputer. The first task, therefore, was to transport picture data to the intended Picture System host (a North Star "Advantage" microcomputer). A Fortran program QTRAN was written to translate the source files into a display ordered list of quads and these files were subsequently loaded via an RS232

serial link to the host and held as CP/M disk files on minifloppies. In order to avoid data corruption at this critical stage the DOLs were coded in ASCII characters allowing parity checking on transfer. These files form the basis for the other disk based data structures which have evolved. Every picture thus exists as a variety of disk files which, following the CP/M convention, are designated ‹PICTURE.*› where PICTURE is a unique picture name and * represents a three letter extension word denoting the file type. The various file structures will now be described.

PICTURE.PIC files are the original printable ASCII display ordered lists and are structured thus:

```
:nnccsccsccsccs........ccsccs(CR)(LF)
:nnccsccs.....................(CR)(LF)

:00
```

where nn = number of quad records in the line expressed in hex

cc = 8 bit colour code in hex

s = 4 bit size code

```
0 = even field pixel
8 = odd field pixel
1 = 2 x 2 pixels
2 = 4 x 4
3 = 8 x 8
4 = 16 x 16
5 = 32 x 32
6 = 64 x 64
```

PICTURE.BPF files are binary translations of the ASCII files with two bytes allocated to each quad and a two byte header and EOF marker thus:

```
23 FF cc cs cc cs ..... cc cs FF FF
```

Notice the colour code is now allocated twelve bits although only eight are used. The four bit size code in these files corresponds

to that in table 4.1.

PICTURE.ZPF files are binary zoned display ordered lists with the format:

```
zz FF hh hh cc cs cc cs ..... hh hh .....hh hh ....
...... zz FF ........... zz FF ........... FF FF
```

where zz FF is a two byte zone header with zz = zone number

hh hh is a two byte scanline header (as in 6.2.2)

FF FF is the end of file marker.

PICTURE.ZTF files are binary zoned quadtree files with the format:

```
7F zz xx xx xx xx xx xx ......... xx xx 7F zz ....
......... 7F zz ......... 7F zz ............. 7F 00
```

where 7F zz is a two byte zone header with zz = zone number

7F 00 is a two byte end of file marker

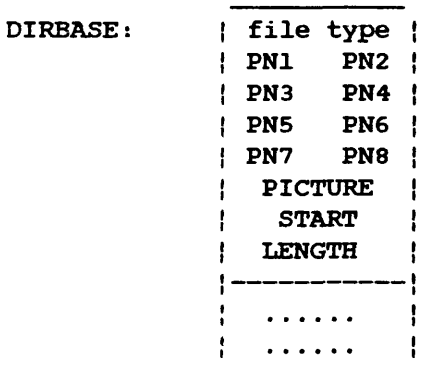
xx xx is a two byte quadrant code (as in 6.2.3)

## 6.4 Communication with the host

Picture files stored on on mini-floppy disk may be downloaded to the display processor via an RS232C serial line or a Centronics style parallel link. The host runs a file server program TFS which monitors both communication links for file requests and responds by transmitting the appropriate file (or "?" if the file is not available).

## 6.5 The Picture Directory

Access to locally stored pictures is provided by the maintenance of a picture directory with a maximum of sixteen entries. The directory format is as follows:




where file type = 0 indicates a vacant directory entry
= 1 indicates a display ordered list
= 2 indicates a zoned display ordered list
= 3 indicates a zoned quadtree
PN1-8 = ASCII picture name characters
PICTURE_START = 32 bit pointer to picture file
LENGTH = 16 bit file length in words

## 6.6 Picture System Firmware

The system firmware is written entirely in 68000 assembly language and is supported by a simple monitor [42] which provides various utilities and debugging facilities. All assembled code was committed directly to EPROM and debugged in situ with the aid of the monitor.

## 6.7 System commands

The system responds to a variety of commands typed at the terminal keyboard. Only the first two letters of each command word are significant though the entire word may be used. Some commands require an argument PICTURE which is a picture name using up to eight characters, the first of which must be alphabetic. The commands are summarised and then individually described below.

```
list

clear

erase PICTURE

mode

load PICTURE

display PICTURE

screen

exit
```

### 6.7.1 List

This command causes listing at the terminal of the names of pictures held locally by searching the directory for non-zero file types.

### 6.7.2 Clear

This command clears the screen by sending to the picture buffer a display list consisting of sixty four 64 x 64 pixel black quads.

### 6.7.3 Erase PICTURE

This command interrogates the picture directory and alters the file type of the appropriate entry to zero. There is no attempt at a sophisticated memory management scheme and the remaining files are simply compacted together so that new files are always loaded into the first free space. Since both variants of the DOL demand consecutive storage this is by far the most convenient procedure. However, in a system using only quadtree files a more complex scheme might be considered worthwhile.

### 6.7.4 Mode

Following this command the user is prompted by a sequence of messages to select the format of picture data to be loaded from the host and to choose whether the serial or parallel link is to be activated. The responses are used to set the current mode which is stored as the lower four bits of the word MODEFLG where:

| | 0 | 1 |
|---|---|---|
| bit 0 | serial load | parallel load |
| bit 1 | full-frame | zoned |
| bit 2 | ascii | binary |
| bit 3 | DOL | quadtree |

### 6.7.5 Load PICTURE

The directory is first inspected to ensure that the file is not already loaded and to locate free store. After reading MODEFLG to determine format, a file request is issued to the host on the appropriate channel. This comprises:

*PICTURE(CR) to request a .PIC file

#PICTURE(CR) to request a .BPF file

$PICTURE(CR) to request a .ZPF file

%PICTURE(CR) to request a .ZTF file.

If found at the host, the file is downloaded and the appropriate local RAM file is created. If not found a query character '?' is returned.

### 6.7.6 Display PICTURE

The directory is examined to find the file type and start address of the picture. If the picture is not available locally it

is downloaded from the host. According to the file type the appropriate action is taken to compile a DOL into the picture buffer. In every case the display screen is blanked during compilation to avoid objectionable noise.

Display of a picture stored as a DOL is very straightforward since all that is required is a block move of data to the picture buffer.

Display from a zoned DOL involves merging quad data from eight of the sixty four zone DOLs at a time, thereby creating the frame DOL strip by strip. This is accomplished by using the embedded scanline headers and maintaining an array of eight pointers into the active set of zone DOLs. The processing of each strip is initiated by inspecting, in order, the scanline headers of each active zone for quads starting on scanline zero. These quads are output to the picture buffer. To minimise processing time there is a look-ahead on each pass to determine the scanline key for the next pass.

Figure 6.2 shows that the display ordered traversal of a quadtree is by no means straightforward. Bearing in mind the fact that the majority of quad data lies in the lower levels of the tree it is clear that the time spent following pointers must be minimised by designing a traversal algorithm which visits each node just once. As will be shown this can be achieved at the expense of comparatively modest buffering using a single stack and five queues, one corresponding to each level in the tree apart from the zone roots and pixel level. The traversal algorithm is presented on

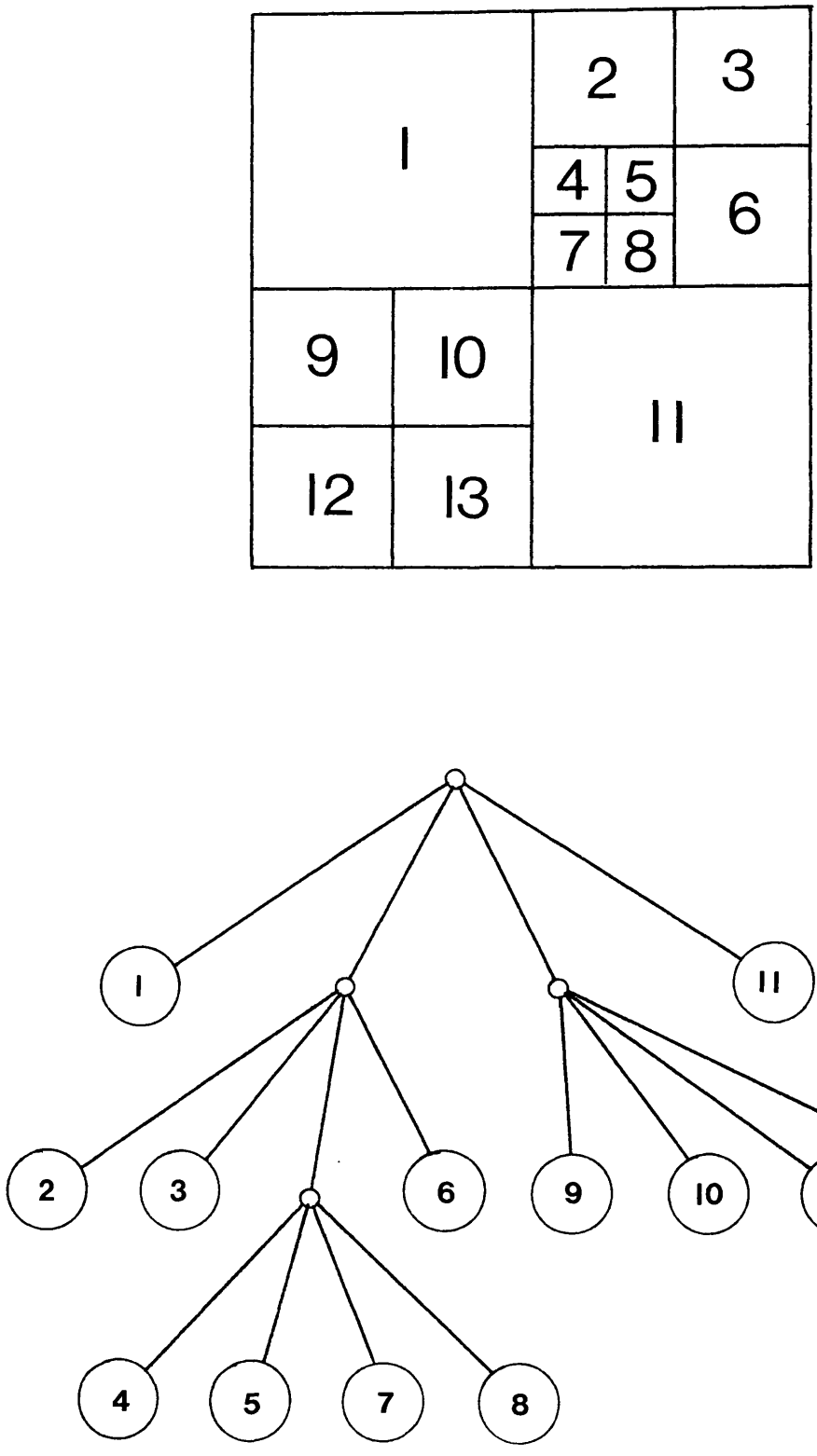



Figure 6.2 Display order traversal

the following page and may be examined to determine the quantity of storage required by the various buffers in the worst case of the tree extending in every branch down to pixel level. Queue(6) is used as buffer storage for size 6 quadrant codes and might need to store two words from each of the eight zones active at any time ie. 16 words. Each of these in turn may generate two words for queue(5) which therefore requires 32 words and so on down to queue(2). Accordingly, the amount of storage needed is as follows:

| | |
|---|---|
| queue(6) | 16 words |
| queue(5) | 32 words |
| queue(4) | 64 words |
| queue(3) | 128 words |
| queue(2) | 256 words |

The greatest demand for stack space would occur if the whole of queue(2) were transferred to it, which including size codes implies a maximum storage of 512 words.

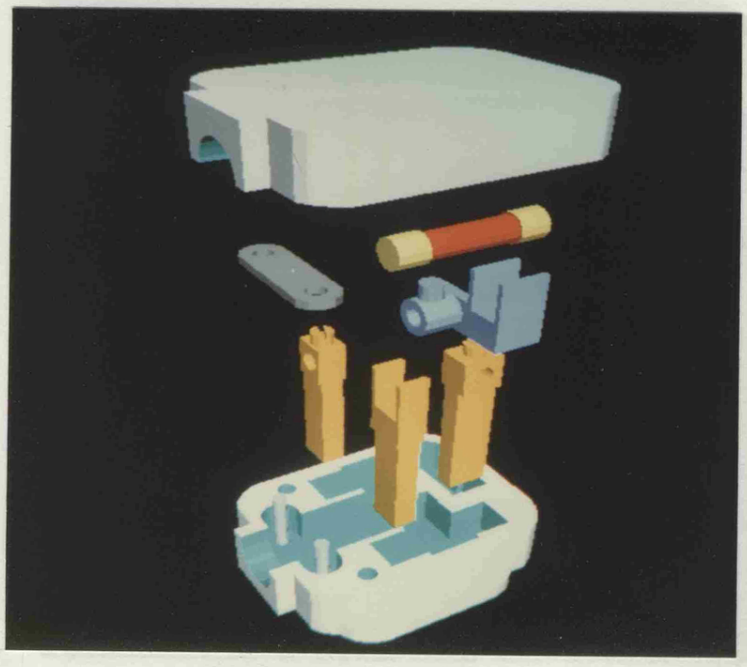
### 6.7.7 Screen

This command was implemented in the latter stages of the project to assess the flexibility of the zoned quadtree data structure and may only be used when a picture is displayed from this structure. In screen mode it is possible to perform a limited range of pan and zoom operations under the control of key depressions at the terminal. Each zoom-in causes twofold magnification and may be continued until a single picture zone occupies the whole screen ie. eightfold magnification. In each case the zoom acts on picture content at the centre of the screen when

```
procedure to compile DOL from zoned quadtree
begin
clear the buffer counters;
for strip:= 1 to 8 do
     begin
     size:= 7;
     for the first/next 8 words from the list of zone quadrants do
          begin
          push quadrant code on to stack;
          push size on to stack
          end;
     repeat
          size:= pop stack;
          quadrant code:= pop stack;
          if quadrant code tag indicates a leaf then
               begin
               get its colour;
               pack it together with the current size code into a
               single leafcode word;
               move it to the next picture buffer location
               end
          else
               begin
               {the quadrant code must be an offset so use this to
               access a new node}
               size:= size - 1;
               if size=1 then {the node quadrants must be pixels}
                    begin
                    send appropriate pixel leafcodes in the correct
                    order to the picture buffer;
                    end
               else {buffer up the node quadrant codes}
                    begin
                    push NEquadrant code;
                    push size;
                    push NWquadrant code;
                    push size;
                    add SWquadrant code to queue(size);
                    add SEquadrant code to queue(size);
                    end
               end;
          if stack is empty then
               begin
               starting with queue(2) search the queues for
               quadrant data;
               take all the data from the first occupied
               queue and transfer to the stack in reverse
               order together with size values
               end
     until the stack is empty
     end {of processing one strip}
end.
```

possible. In any of the magnified formats it is possible to pan vertically or horizontally with the restriction that all displayed images must be zone aligned. Figure 6.3 illustrates a sequence of operations in screen mode and shows how the displayed image relates to the zone matrix. It is important to understand that in the present system the "magnified" images are not displayed at the resolution of the original; in fact at full zoom each pixel in the original has become an 8 x 8 quad so the effective resolution is reduced to 64 x 64 ie. the pixel resolution of a single zone. The effect is therefore equivalent to zooming by pixel replication in a conventional framestore display. In Chapter 8 an extension of the principles employed in screen mode are shown to be applicable to a display processor which retains full resolution at several levels of zoom.

Execution of screen mode operations is made possible by very minor modification of the zoned quadtree display algorithm which has already been described (6.7.6). To illustrate the principle let us consider the display of a 2 x 2 array of zones from the picture zone matrix. The picture consists of two strips each with two zones so processing of each strip is initialised by pushing the appropriate two quadrant codes from the zone list on to the stack. Since each zone will now cover an area of 256 x 256 pixels on the screen the size must be set equal to 9 instead of 7. The possibility now arises that when a leaf quadrant is drawn from the stack it may have a size greater than 7. If so, it cannot be sent to the picture buffer but must be decomposed into smaller quads.



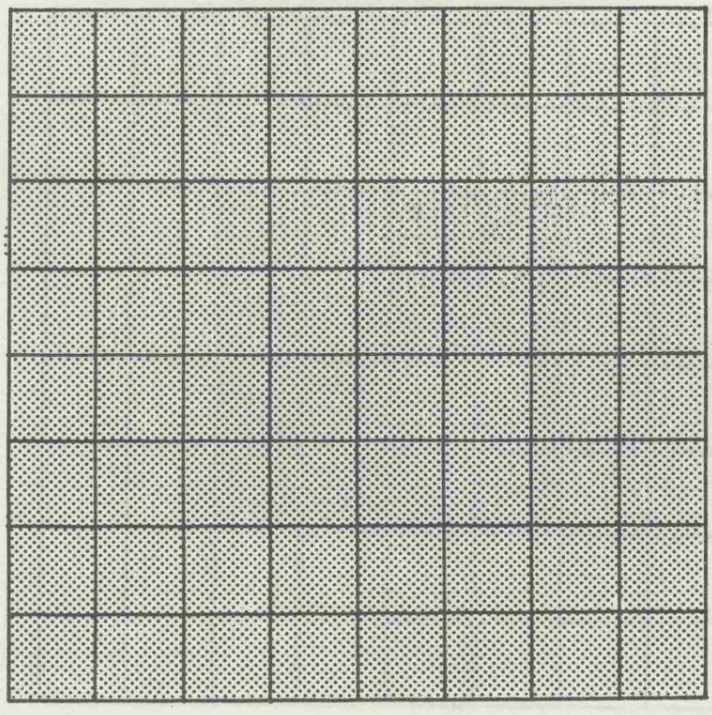
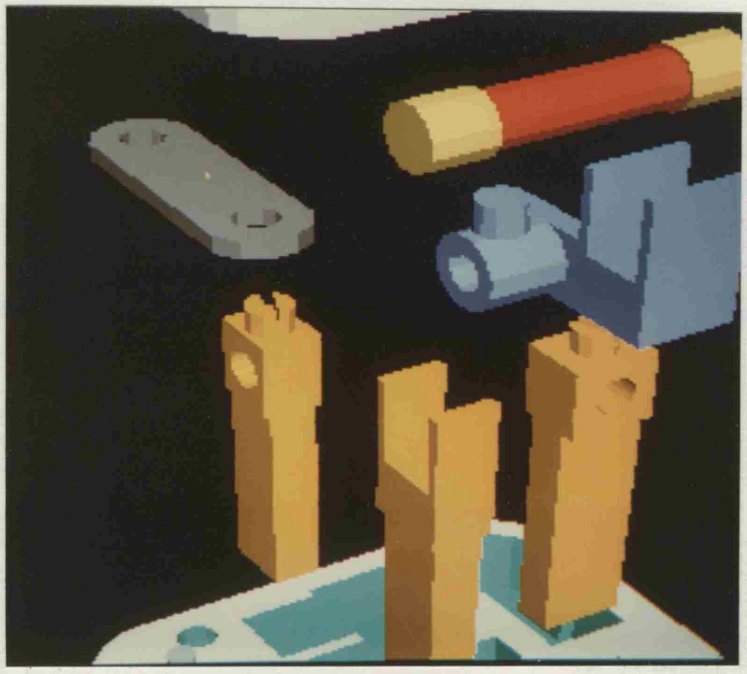


Figure 6.3a The complete picture



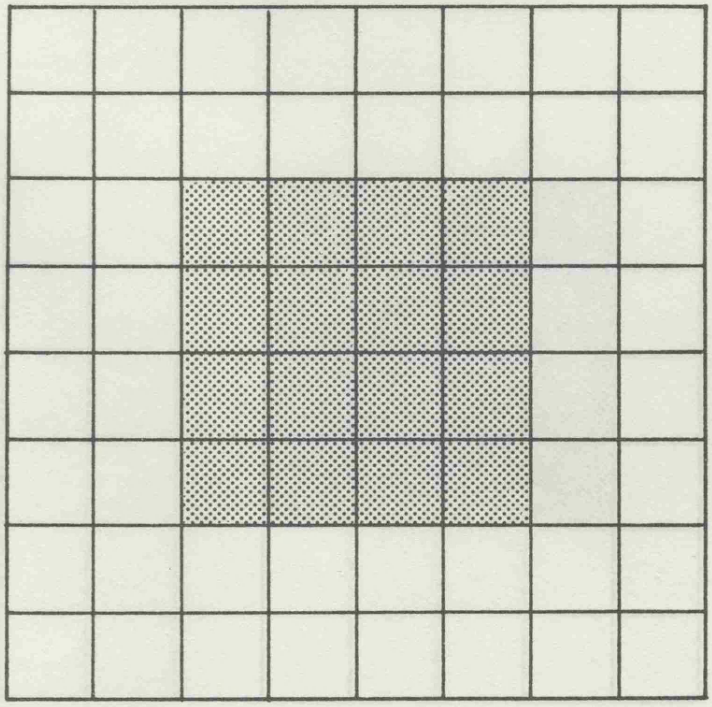
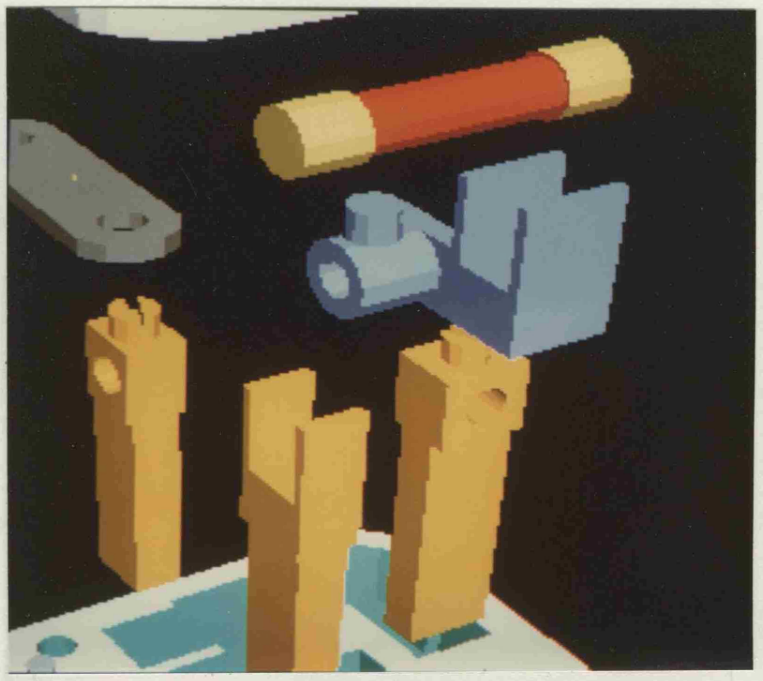


Figure 6.3b Zoom in



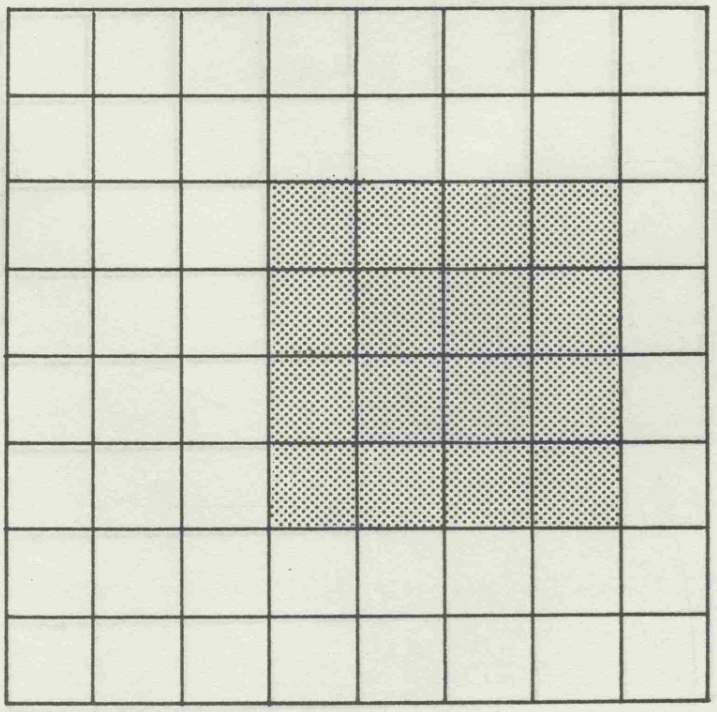
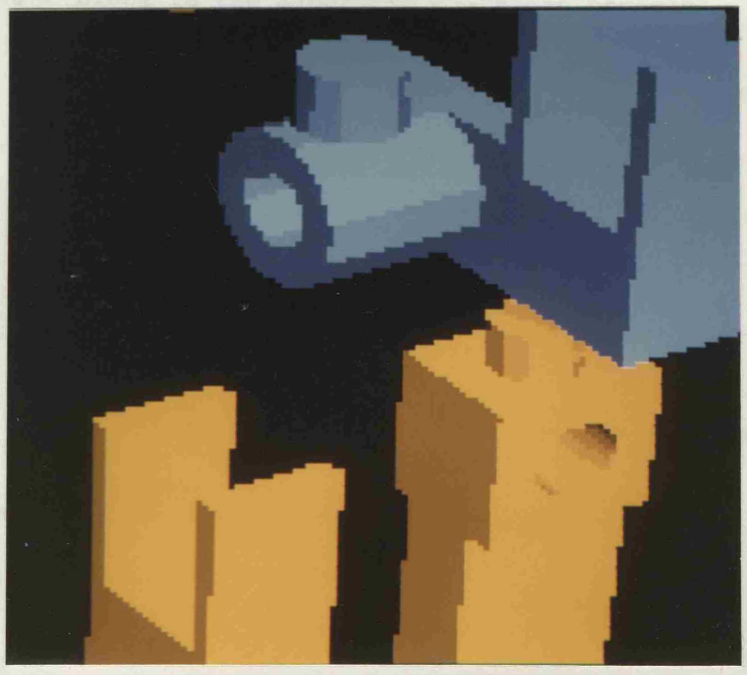


Figure 6.3c Pan right



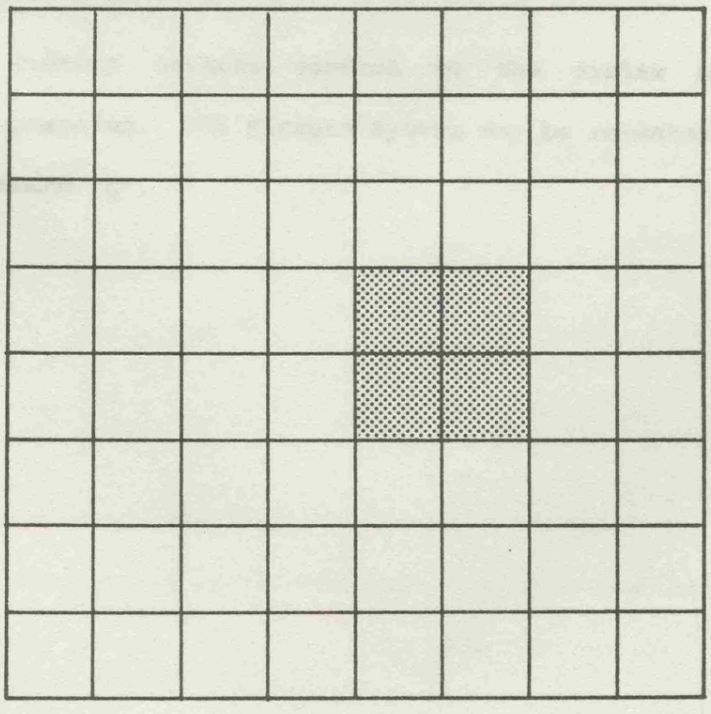


Figure 6.3d Zoom in again

The relevant block in the algorithm therefore becomes:

```
if quadrant code tag indicates a leaf then
     begin
     if size > 7 then
          begin
          size:= size - 1;
          push quadrant code;
          push size;
          push quadrant code;
          push size;
          add quadrant code to queue(size);
          add quadrant code to queue(size)
          end
     else generate leafcode and send to picture buffer
     end;
```

With three stages of zoom provision must be made for extra buffering:

queue(9) 2 words

queue(8) 4 words

queue(7) 8 words

### 6.7.8 Exit

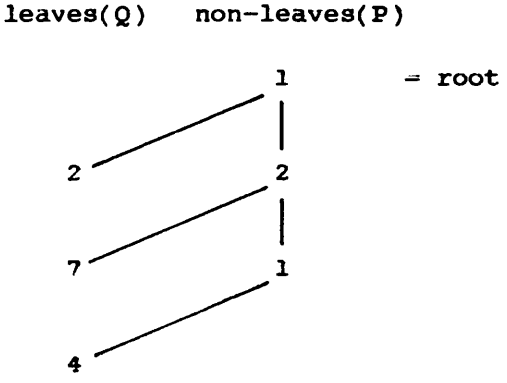
This command returns control to the system monitor for debugging purposes. The Picture System may be re-entered with the monitor command "Q".

## Chapter 7 QUADTREE PICTURE STATISTICS AND SYSTEM PERFORMANCE

Analysis of the display ordered list for a picture provides information on the distribution of leaves in the quadtree representation. It will become clear that there is considerable consistency in these distributions throughout the lower levels of the tree and this suggests a certain invariance which might have important consequences for picture coding. The chapter concludes with a summary of the picture system performance which may be related to the picture statistics and perceived complexity of the displayed pictures (Photographs 1 - 5).

### 7.1 Quadtree leaf distribution

The quadtree statistics are most easily presented as a binary tree showing the number of leaves and non-leaf nodes at each level in the tree. For example, figure 3.2 would be described by




The statistics for five pictures are presented in table 7.1 below. It should be remembered that the trees employed in this picture system branch sixty-four ways from the root.

| | tap | | lever | | plug | | it | | engine | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Q | P | Q | P | Q | P | Q | P | Q | P |
| 64 x 64 | 31 | 33 | 39 | 25 | 29 | 35 | 15 | 49 | 23 | 41 |
| 32 x 32 | 40 | 92 | 16 | 84 | 39 | 101 | 50 | 146 | 29 | 135 |
| 16 x 16 | 135 | 233 | 92 | 244 | 121 | 283 | 205 | 379 | 144 | 396 |
| 8 x 8 | 438 | 494 | 395 | 581 | 459 | 673 | 661 | 855 | 568 | 1016 |
| 4 x 4 | 1072 | 904 | 1198 | 1126 | 1310 | 1382 | 1808 | 1612 | 2028 | 2036 |
| 2 x 2 | 2334 | 1282 | 2898 | 1606 | 3531 | 1997 | 4217 | 2231 | 5282 | 2862 |
| 1 x 1 | 5128 | -- | 6424 | -- | 7988 | -- | 8924 | -- | 11448 | -- |
| Totals | | | | | | | | | | |
| Q | 9178 | | 11062 | | 13477 | | 15880 | | 19522 | |
| P | 3038 | | 3666 | | 4471 | | 5272 | | 6486 | |
| P+Q | 12216 | | 14728 | | 17948 | | 21152 | | 26008 | |
| Q/(P+Q) | 0.751 | | 0.751 | | 0.751 | | 0.751 | | 0.751 | |
| (1x1)/Q | 0.56 | | 0.58 | | 0.59 | | 0.56 | | 0.59 | |
| (2x2)/Q | 0.25 | | 0.26 | | 0.26 | | 0.27 | | 0.27 | |
| (4x4)/Q | 0.12 | | 0.11 | | 0.10 | | 0.11 | | 0.10 | |
| (8x8)/Q | 0.05 | | 0.04 | | 0.03 | | 0.04 | | 0.03 | |

Table 7.1

Several important features emerge from the table.

a) The proportion of leaf nodes expressed as a fraction of the total number of nodes ie. Q/(P+Q) is constant for each picture. This supports the contention (3.2.1) that the pointers used to construct a minimal linked tree comprise 25% of the picture data.

b) The proportion of leaves at each level in the lower part of the tree is reasonably constant for each picture. Unfortunately, without analysing quadtree data for other picture types (eg. line drawings and text) it is impossible to say whether the distribution is invariant. Oliver and Wiseman [38], whose investigation included line drawings, support the value of 75% for Q/(P+Q) so it is reasonable to speculate that their other statistics would be similar to those presented here.

c) 93% or more of the picture quads are located in the lowest three levels of the quadtree and about 98% are in the lowest four levels. These figures justify the decomposition of large quads in the tree; indeed it might be feasible to work with picture zones much smaller than 64 x 64 pixels. The panning process described in 6.7.7 could then be made smoother but the rich connectivity which is a feature of the quadtree would be further weakened.

## 7.2 Generalised quadtree statistics

It has been observed that the distribution of nodes in the quadtree representation of several pictures is reasonably constant. A brief investigation of this property is worthwhile since it will prove helpful in predicting the extent to which a quadtree grows when a given picture is displayed at increased resolution.

The quadtree growth is best understood by applying some of the basic results from the field of stochastic geometry. The proofs of these results may be located in most texts on the subject (see for example [43]) and will not be repeated here. It may be shown that when a square is divided regularly into N by N equal cells and a random straight line is drawn through the square, it is expected that it will intersect on average N cells. This result can be applied with a high degree of accuracy in the lower levels of the quadtree since the boundaries between coherent picture areas can be approximated to straight line segments over a small region and are likely to be randomised in position and orientation. Let us start, for example, by assuming that a boundary cuts through a size 16 by 16 quadrant. On average this will leave two 8 x 8 quads intact and

intersect two others. These in turn each decompose into two 4 x 4 quads and so on producing:

| | Q | P |
|---|---|---|
| 16 x 16 | ? | 1 |
| 8 x 8 | 2 | 2 |
| 4 x 4 | 4 | 4 |
| 2 x 2 | 8 | 8 |
| 1 x 1 | 32 | - |

| | | |
|---|---|---|
| Totals: | Q | = 46 |
| | P | = 15 |
| | P+Q | = 61 |
| | Q/(P+Q) | = 0.754 |
| | (1x1)/Q | = 0.70 |
| | (2x2)/Q | = 0.17 |
| | (4x4)/Q | = 0.09 |
| | (8x8)/Q | = 0.04 |

Comparison with the figures obtained in practice show that the ratio Q/(P+Q) is in reasonable agreement. However, the proportion of pixel size quads is too high whilst the proportions of the larger quads are too small. This discrepancy may be explained by examining the tree at its final subdivision. Wherever a boundary passes through a 2 x 2 quadrant a decision must be made regarding the colouring of each of the four pixels. If an individual pixel lies entirely inside one region then it may safely be assigned the colour of that region; if, on the other hand, the boundary cuts a pixel then its colour may be assigned in one of two ways.

a) The simplest method is to test in which region the centre of the pixel lies and then to assign the appropriate colour to that pixel (fig.7.1a). This is the strategy which has been adopted in the generation of the sample pictures. As a consequence some of the "subdivided" 2 x 2 quadrants will end up with all four descendant pixels the same colour, in other words, as coherent 2 x 2 areas.

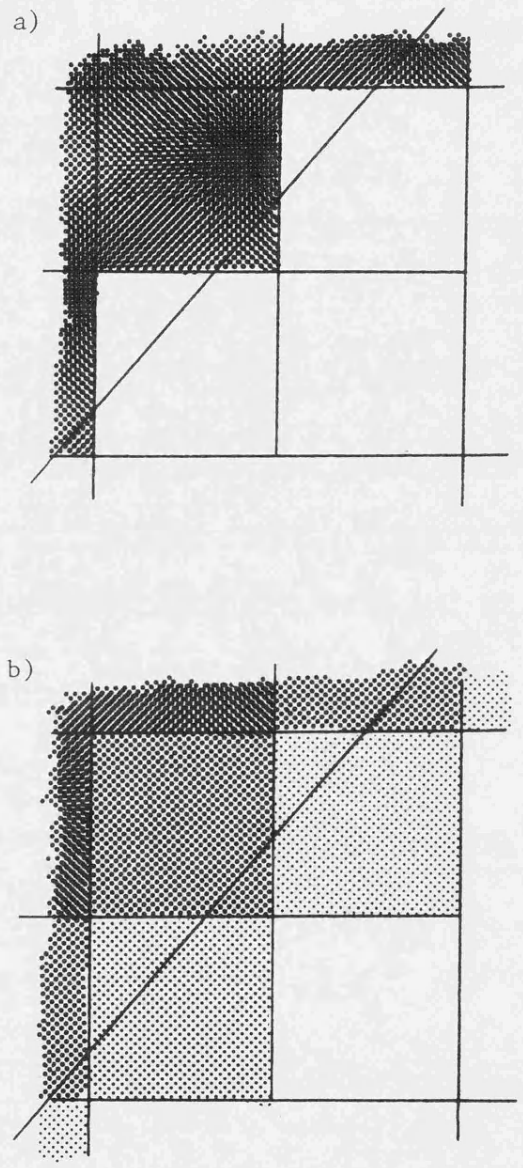



Figure 7.1 Allocating pixel colour

This has the effect of reducing the proportion of pixels and increasing the proportion of 2 x 2 quads. A similar condensation occurs at every level in the tree as the decision at pixel level propagates upwards. The statistical discrepancy is therefore justified in qualitative terms although no quantitative analysis has been performed.

b) A superior method of assigning pixel colour has the benefit of removing the jagged edges (aliasing) caused by the previous strategy. The distance of each pixel centre from the notional boundary can be used to calculate a weighted average of the opposing colours (fig.7.1b). This has the effect of providing a visually smooth delineation of the separate regions. It may therefore be predicted that pictures which have been anti-aliased will conform to the statistical model presented earlier in this section.

Adopting the simple model of quadtree growth it is easy to predict the consequences of increasing the display resolution by a factor of two. The tree becomes:

| Q | P |
|---|---|
| ? | 1 |
| 2 | 2 |
| 4 | 4 |
| 8 | 8 |
| 16 | 16 |
| 64 | -- |

Totals: Q = 94
P = 31
P+Q = 125

and the tree has doubled its size. Notice the ratios remain essentially as before.

## 7.3 Local picture file statistics

The three varieties of local picture data structure have been described in 7.2. The storage (in words) required for each of the five sample pictures is presented in table 7.2.

| | DOL | zoned DOL | zoned quadtree |
|---|---|---|---|
| tap | 9178 | 10040 | 12216 |
| lever | 11062 | 11894 | 14728 |
| plug | 13477 | 14440 | 17948 |
| it | 15880 | 17170 | 21152 |
| engine | 19522 | 20744 | 26008 |

Table 7.2

As might be expected the figures for the DOL and the zoned quadtree are equal to Q and P+Q respectively so the zoned quadtree files are 33% longer than the DOLs. The zoned DOLs impose a smaller overhead which varies from about 9% for "tap" to about 6% for "engine".

## 7.4 Loading pictures from the host

Picture data may be downloaded from the host using either the serial line (operating at 19200 baud) or the parallel link. The load times for each of the five sample pictures in their different formats are presented in table 7.3. The figures include disk latency and are given in seconds to the nearest half second.

| | serial link | | | parallel link | | |
|---|---|---|---|---|---|---|
| | DOL | zoned DOL | zoned quadtree | DOL | zoned DOL | zoned quadtree |
| tap | 19.0 | 19.5 | 23.5 | 5.5 | 6.0 | 6.5 |
| lever | 22.5 | 23.5 | 29.5 | 6.5 | 7.5 | 8.5 |
| plug | 26.0 | 28.0 | 35.5 | 7.5 | 8.5 | 10.5 |
| it | 31.5 | 33.5 | 41.5 | 9.0 | 10.0 | 12.0 |
| engine | 39.0 | 40.5 | 51.0 | 11.5 | 12.0 | 14.0 |

Table 7.3

By comparison, a full 12 bit colour pixel representation would require

512 x 512 x 12 / 16 = 200 000 words of storage

which is an order of magnitude greater than quad encoded pictures of the complexity illustrated. The transmission time for pictures in full bit-mapped format would consequently be approximately 400 seconds under the same conditions of serial transmission.

## 7.5 Display of locally stored pictures

The delay from requesting a picture to its display on the screen depends upon the amount of processing necessitated by the local storage format. Table 7.4 quantifies the latency in the display from each data structure. Figures were obtained by timing a multiple execution of the appropriate display routine.

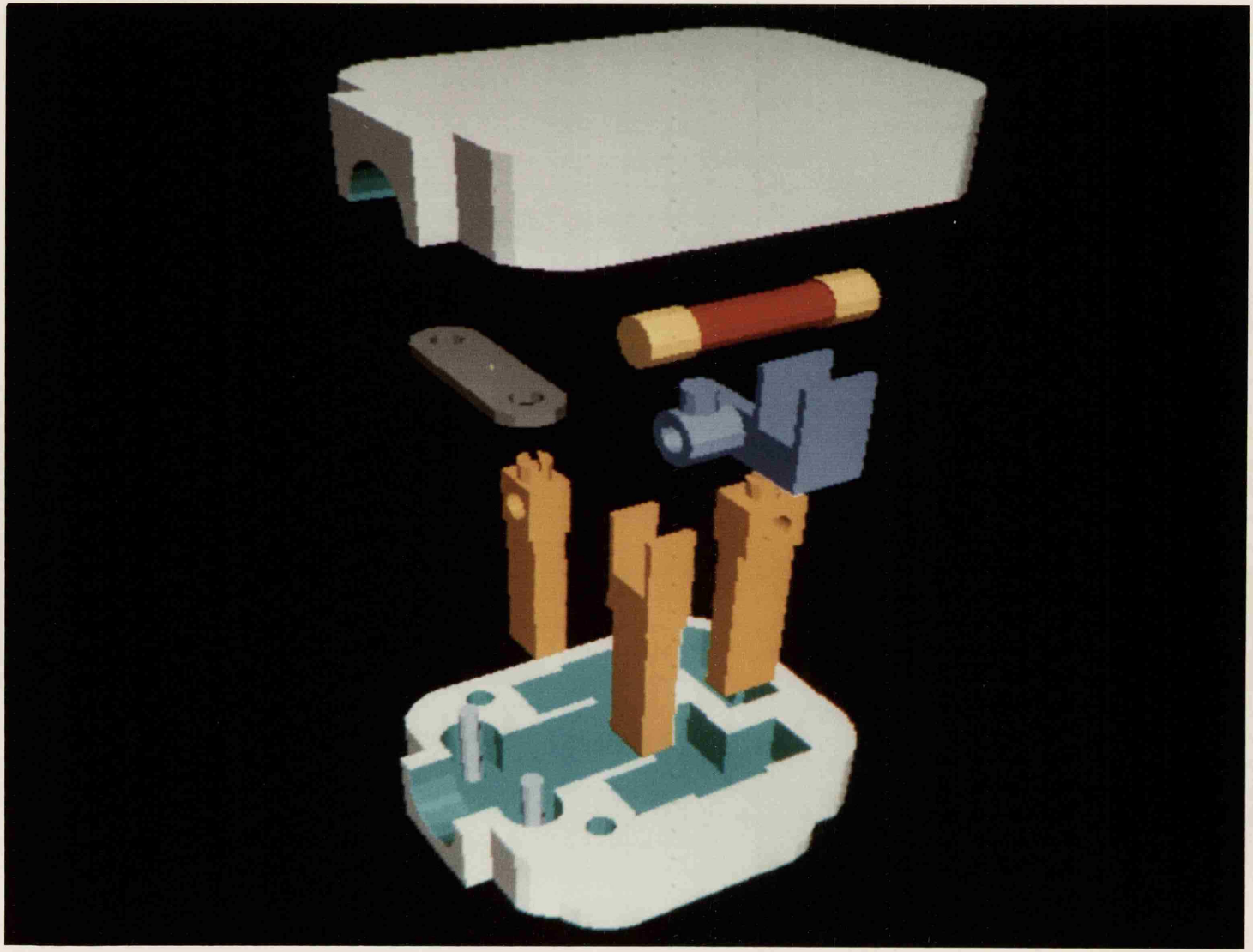
| | DOL | zoned DOL | zoned quadtree |
|---|---|---|---|
| tap | 15ms | 75ms | 250ms |
| lever | 18ms | 80ms | 290ms |
| plug | 22ms | 100ms | 350ms |
| it | 26ms | 105ms | 420ms |
| engine | 32ms | 120ms | 500ms |

Table 7.4

Display from both types of DOL is clearly very rapid. The compilation times from the linked quadtree structure indicate an average delay of about 26ms per 1000 quads.

## Photographs

1. Tap
2. Lever
3. Plug
4. IT
5. Engine



# Chapter 8 PROPOSALS FOR APPLICATION AND FUTURE INVESTIGATION

It has been the intention of this thesis to demonstrate that quadtree picture code offers considerable potential as a display source. Since the present system is essentially a research vehicle it is important to consider how it might best be developed. The chapter starts by examining the extent to which quadtree code might be used for interactive computer graphics. Following this, two applications are suggested as a consequence of the results obtained during the present work. The first proposal describes a coding scheme for the transmission of images. The second application is a picture archive system which is a direct development of the current display processor.

## 8.1 Interaction with quad encoded pictures

Quadtree picture code, like other compression strategies, suffers a major drawback: it does not easily lend itself to fully interactive graphics applications. Considering the virtues of quadtree code in other respects it appears that further research into its interactive potential is worthwhile.

The conceptually simple operation of altering a single pixel value expands into a sizeable task when a picture is stored as compressed code. One option (which applies to all code types) consists in expanding the relevant section of code into a full pixel representation, modifying this representation and finally re-encoding. Since the new code is unlikely to fit neatly into the space it originally occupied there is an additional problem of

memory management. However, a second option is available in the case of a picture stored as a linked quadtree. Given the screen coordinates of a pixel there is a simple transformation which yields the route through the quadtree (Appendix 1). If the target pixel is part of a larger quad then that quad must be decomposed recursively. Operations on single pixels may therefore be accomplished reasonably efficiently but line drawing, though it may be regarded as a succession of pixel operations, may prove very slow. As the length of line increases there will come a breakeven point beyond which it becomes faster to expand a whole zone, draw the line and then re-encode.

In contrast to the problems of line drawing, one operation that is featured in most computer graphics packages is likely to be much faster using quadtree code, namely area fill. The identification of connected regions is a very familiar task in image processing and the value of quadtrees in this discipline is well proven.

## 8.2 A coding scheme for the transmission of images

The survey of picture coding presented in Chapter 2 mentions briefly the use of variable length codewords (2.5.7) for exploiting statistical properties of coding schemes. Unfortunately the statistics for many coding strategies will vary according to the exact nature of the picture. An important feature of quadtree coding is that the statistics appear to be invariant over a wide range of pictures (see Chapter 7). This suggests that variable length code may be effectively applied to the transmission of

pictures represented as quadtrees.

The following analysis is restricted to images in which it is assumed that each colour is equally likely. Limiting the maximum size of quad to 16 x 16 will not alter the leaf distribution significantly so it is assumed that the frequency of codewords remains roughly

1 x 1 .... 58% (codeword A1)

2 x 2 .... 26% (codeword A2)

4 x 4 .... 11% (codeword A3)

8 x 8 .... 4% (codeword A4)

16 x 16 ... 1% (codeword A5)

The source entropy H for this group of codewords is calculated by the formula:

$$H = -\sum_i p(Ai) \log_2 p(Ai)$$

$$= 1.56 \text{ bits per codeword}$$

By assigning the binary codes as follows

A1 = 0 L(A1) = 1

A2 = 10 L(A2) = 2

A3 = 110 L(A3) = 3

A4 = 1110 L(A4) = 4

A5 = 1111 L(A5) = 4

the average bit length L is calculated as

$$L = \sum_i p(Ai) L(Ai)$$

$$= 1.63 \text{ bits per codeword}$$

giving a coding efficiency of about 96%.

In practice the picture would be transmitted as a display ordered list with each size codeword preceding an N bit colour code. As a simple example consider the transmission of a binary image (N = 1) where white = 0 and black = 1. The display ordered list of quads: 4x4 white, 2x2 white, 2x2 black, 1x1 white, 1x1 black would be represented by the bit-stream (1100) (100) (101) (00) (01). The parentheses are included here only to aid identification of the separate items and are not part of the code. It should be noted that the context is always clear if the code is transmitted error-free.

Since each quad requires an average of (1.63 + N) bits it is possible to determine the compression ratio for pictures of various complexity. For example, a picture which decomposes into 10000 quads at a resolution of 512 x 512 pixels obtains a compression ratio of

$$\frac{512 \times 512 \times N}{10000 \times (1.63 + N)} \approx 26.2 \times \frac{N}{1.63 + N}$$

This expression evaluates to approximately 10 for N = 1 and has an upper limit of about 26 as N approaches infinity. If the resolution is increased by a factor of two then the number of quads roughly doubles so the compression ratio is increased by a factor of two.

Kawaguchi and Endo [34] report compression ratios obtained using a different method of quadtree encoding (3.2.3) for six binary images at a resolution of 1024 x 1024 pixels. Fortunately their paper includes quad totals for each picture so a comparison with the proposed coding scheme is possible. In every case the

compression ratio predicted for the proposed code is very similar.

## 8.3 The display of large pictures

The proposal which is outlined in this section is for a picture archive system which allows the user to request a picture and then zoom into it repeatedly to reveal additional detail. At each stage of magnification there will be a facility to roam incrementally around the picture. Work on this project has been granted support by the Science and Engineering Research Council.

### 8.3.1 The scheme

The proposed system requires an elaboration of the "screen mode" features described in Chapter 6 and demands that the display is maintained at full resolution throughout. This necessitates a large quadtree or rather, since the picture is zoned, an array of quadtrees. Consider, for example, a large picture which is defined to a resolution of 4K by 4K pixels. This may be treated as an array of 64 x 64 zones each comprising 64 x 64 pixels. The picture can therefore be stored as an array of 4096 quadtrees, one for each zone (fig. 8.1). It will shortly become clear that the linked quadtree which has been described previously (3.2.1 and 6.2.3) is inadequate for the proposed scheme since the tree will not be examined to its full depth in many instances. A modified structure is required in which each node contains an extra data field describing the average of its four quadrant colours (fig. 8.2). The problem of colour averaging is discussed in Appendix 2, section 7.

Suppose then, that the entire picture is to be displayed at a

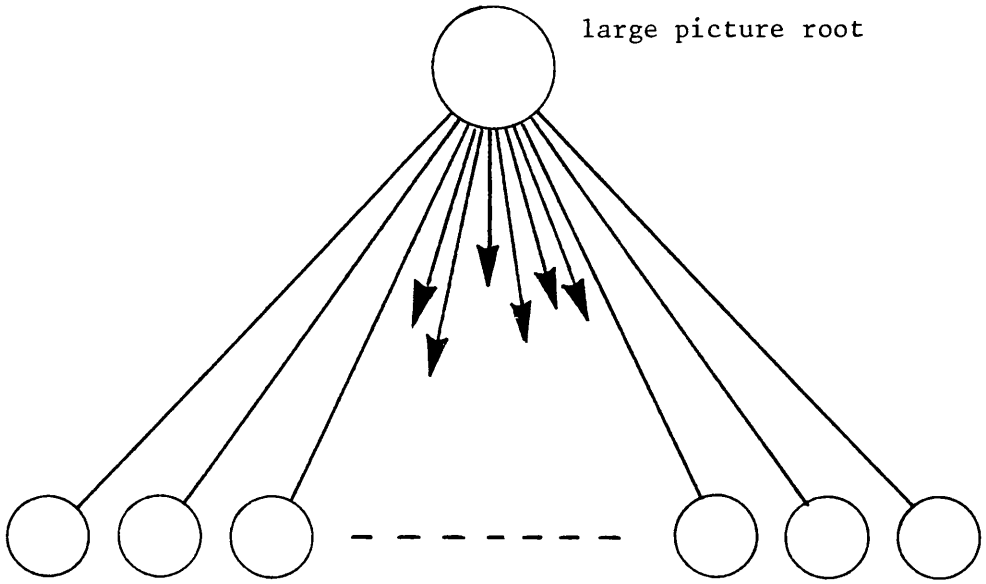



Figure 8.1 Zoning for a large picture

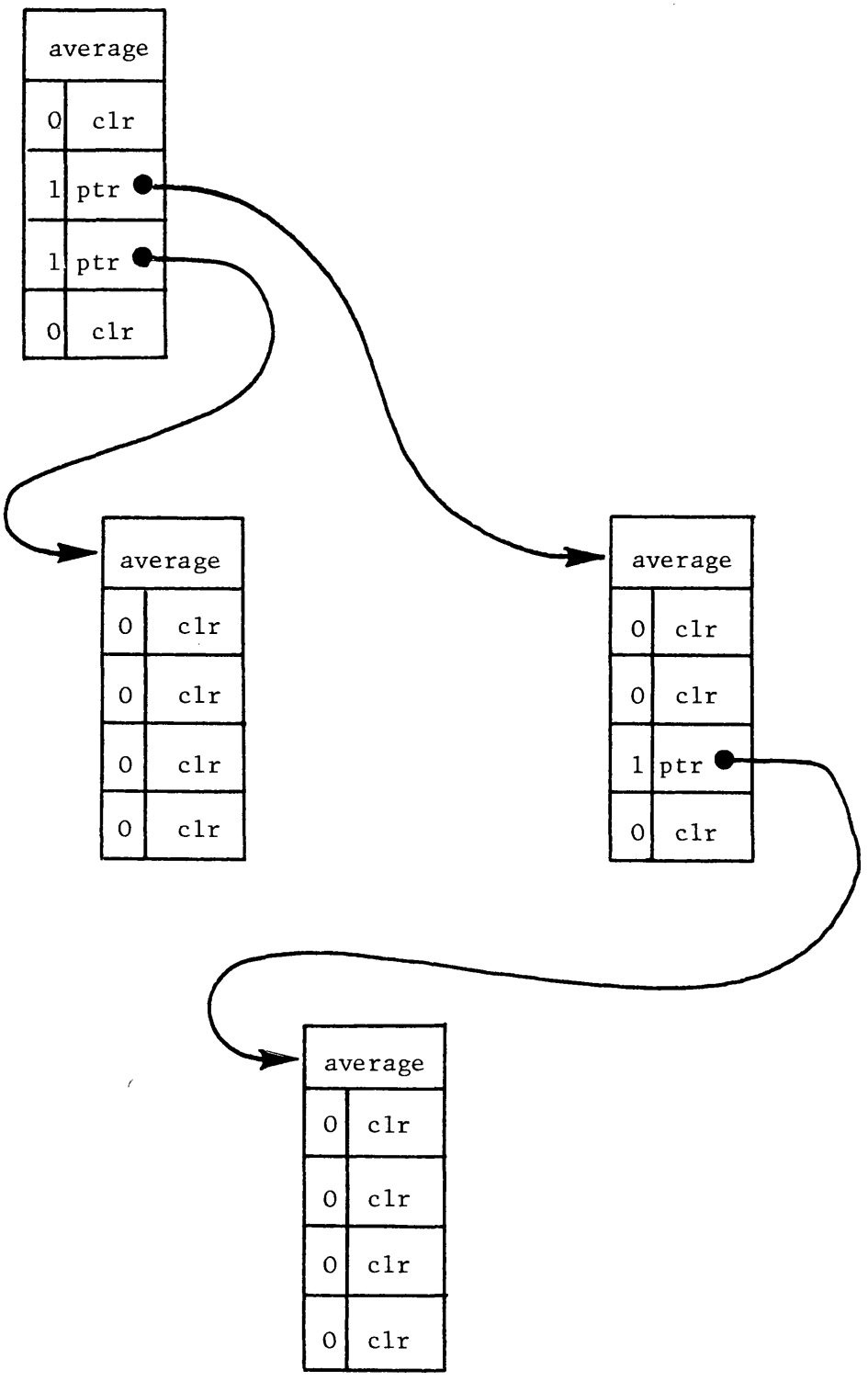



Figure 8.2 Modified linked quadtree structure

resolution of 512 x 512 pixels. All 4096 picture zones are used and each one maps to a screen area of 8 x 8 pixels. The quadtree for each zone therefore needs to be descended by only three levels for the screen resolution to be satisfied. The display processor is programmed to access the node average colour whenever it reaches screen pixel level and this has the fortunate consequence of automatic picture anti-aliasing.

Following the first zoom command the display processor selects a subset of 1024 zones (windowing a quarter of the picture) and follows each quadtree down through four levels because each zone now covers 16 x 16 screen pixels. Two further zoom operations can be performed before the quadtrees are accessed to their greatest depth. At this point the screen will display a subset of 64 zones.

Roaming across the picture at any stage is achieved by altering the zone subset window. It is interesting to note that the roam increment is one zone regardless of zoom setting. Thus at maximum magnification (x8) the perceived movement of the window is one-eighth of the screen, at the next setting (x4) it is one-sixteenth and so on.

### 8.3.2 A practical system

It should be clear from the previous section that the display at any instant is generated from only a fraction of the complete picture data: when the large picture is viewed as a whole the lowest levels in the quadtrees are not required; when a window on the picture is selected only a subset of the 4096 zones are used. There is consequently no particular need for the entire data

structure to be stored in the picture display processor. The majority of data may be held in a host database linked to the display processor. For ultra-rapid picture retrieval the two machines should be closely coupled but for most purposes it is likely that a local area network would provide sufficiently fast access. This option would not be feasible were it not for the structured data compression.

The prototype system which is described in this thesis embodies proven hardware and display algorithms so there remains one main area of study: management of the local and archive data structures. This requires considerable research effort but one desirable feature can be stated immediately. At any stage in its operation the display processor should have cached sufficient data to satisfy the next zoom or pan command. Suppose, for instance, that the screen view is a window on a sixteenth of the picture area (16 x 16 zones). The zone quadtrees will be accessed to a depth of five levels. Therefore, to accommodate a zoom-in request the sixth (and deepest) level of the central 8 x 8 zone quadtrees should also be available for immediate access. In the case of a zoom-out request the quadtrees for the surrounding zones should also be cached but most of these need only have four levels. Only the zones immediately bordering the display area need five levels to accommodate a pan operation. These conditions are summarised in figure 8.3. Clearly then, each command to the display processor invokes compilation of a fresh display ordered list from data which is already held locally. Each operation frees some local memory and

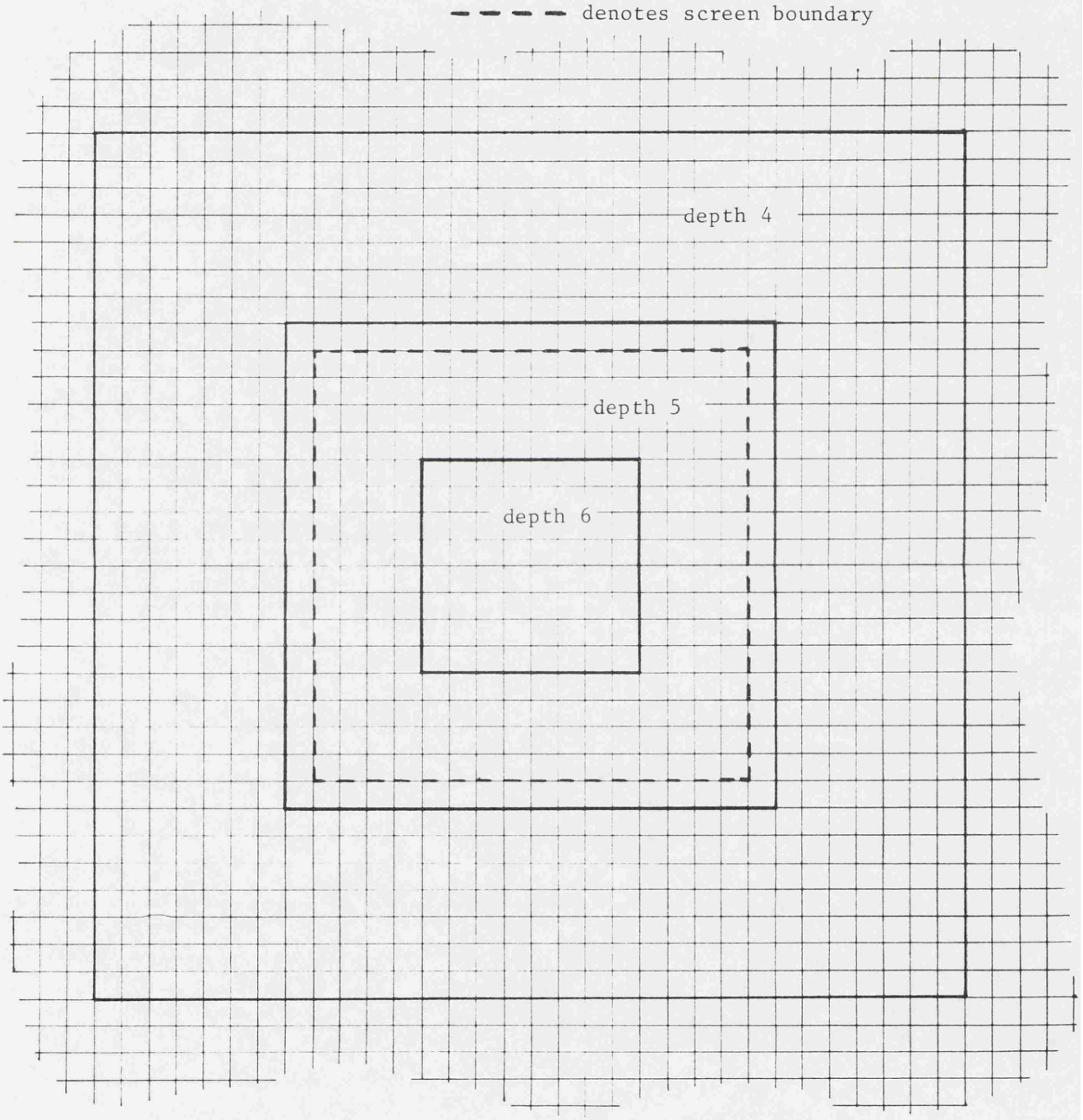



Figure 8.3 Storage requirement at x4 magnification

initiates the downloading of sufficient new data to meet further demand.

### 8.3.3 Storage requirement

To enable a rough estimate of the local storage required by the large picture display processor it will be assumed that a zone tree to a depth of three levels requires storage S. The statistical analysis of section 7.2 suggests that a quadtree doubles in size if it is extended by one level so it can be assumed that:

| quadtree | to | depth | 3 | requires | storage | S |
|---|---|---|---|---|---|---|
| .... | .. | ... | 4 | .... | ... | 2S |
| .... | .. | ... | 5 | .... | ... | 4S |
| .... | .. | ... | 6 | .... | ... | 8S |

Following the prescription for caching described in section 8.3.2 it is possible to estimate the storage requirement at each stage of zoom. Consider again the example illustrated in figure 8.3. There are:

(32x32) - (18x18) = 700 zones at depth 4
requiring 700 x 2S = 1400S units of storage;

(18x18) - (8x8) = 260 zones at depth 5
requiring 260 x 4S = 1040S units of storage;

8x8 = 64 zones at depth 6
requiring 64 x 8S = 512S units of storage.

Total storage 1400S + 1040S + 512S = 2952S units of storage.

The storage at various levels of zoom is summarised in table 8.1 following.

| magnification of screen image | number of zones held at each depth | | | | total storage |
|---|---|---|---|---|---|
| | depth 3 | depth 4 | depth 5 | depth 6 | |
| x 1 | 3072 | 1024 | - | - | 5120 S |
| x 2 | 2940 | 900 | 256 | - | 5764 S |
| x 4 | - | 700 | 260 | 64 | 2952 S |
| x 8 | - | - | 156 | 100 | 1424 S |

Table 8.1

From the figures presented in Chapter 7 a value of S ≈ 70 words (140 bytes if the colour is a 12 bit code) appears generous for pictures of reasonable complexity. The proposed picture system may therefore be implemented with one megabyte of storage.

### 8.3.4 System response

In order to evaluate the response times which might be expected from the proposed system it will be assumed that the display processor is connected to a network with a 1 Megabit/s point-to-point capacity. This is equivalent to about 1000 S per second. The user can therefore expect to wait for about five seconds when a full picture is first called for display. However, if picture data is fetched width first, it will be possible to display a low resolution image almost immediately and with this facility the transmission may be aborted if the requested picture proves to be of no interest to the user. Thereafter, response times can be expected to measure fractions of a second since data will always be locally available. Bearing in mind that data is prefetched in anticipation of further action, the natural interval between commands should generally prove adequate for downloading.

## 8.4 Conclusion

This thesis has demonstrated the feasibility and value of quad encoding as the basis of an integrated approach to the problems of picture storage, transmission and display. The structure imposed by quadtree picture decomposition has been shown to facilitate pan and zoom operations and is well suited to anticipatory caching in the case of large pictures. The scheme promises advantages in the display of high resolution colour pictures where the considerable data compression is a particular virtue in networking.

References


1. Hall, E.L.
Computer Image Processing and Recognition, Academic Press, 1979.

2. Stafford, R.H.
Digital Television: Bandwidth Reduction and Communication Aspects, Wiley, 1980.

3. Webster, D.
Digital picture coding, Wireless World, pp. 67-70, October 1978.

4. Musmann, H.G.
Digital coding of television signals, in Advances in Digital Image Processing (P. Stucki, ed.), pp.61-76, Plenum Press, 1979.

5. Pratt, W.K.
Digital Image Processing, Part 6 - Image Coding, pp.591-731, Wiley, 1978.

6. Levinson, J.Z.
Psychophysics and TV, in Picture Bandwidth Compression (T.S. Huang and O.J. Tretiak, eds.), pp.11-29, Gordon and Breach, 1972.

7. Jordan, B.W. and Barrett, R.C.
A cell organised raster display for line drawings, Communications of the ACM Vol.17, No.2, pp.70-77, February 1974.

8. Kunt, M. and Johnsen, O.
Block coding of graphics: a tutorial review, Proceedings of the IEEE, Vol.68, No.7, pp.770-786, July 1980.

9. Cherry, E.C., Kubba, M.H., Pearson, D.E. and Barton M.P.
An experimental study of the possible bandwidth compression of visual image signals, Proceedings of the IEEE, Vol.51, pp.1507-1517, November 1963.

10. Hunter, G.M.
Full colour television, from the computer, refreshed by run-length codes in main memory, Computer Science Laboratory, Princeton University, Technical Report 182, April, 1975

11. Digital Equipment Co. Ltd.
Logos Colour Display Controller Manual, 1977.

12. Huang, T.S.
Run-length coding and its extensions, in Picture Bandwidth Compression (T.S. Huang and O.J. Tretiak, eds.), pp.231-264, Gordon and Breach, 1972.

13. Daskalakis, C., Laker, R.W. and Hankins, H.C.A.
The application of two dimensional compression schemes to computer graphic display systems, Displays Vol.2, pp.229-235, April 1981.

14. Heaton, A.G., Daskalakis, C, Miles, P.R. and Alvi, M.A.
Data compression in interactive colour graphics using microprocessors, Displays Vol.2, pp.236-241, April 1981.

15. Schreiber, W.F., Huang, T.S. and Tretiak, O.J.
Contour coding of images, in Picture Bandwidth Compression (T.S. Huang and O.J. Tretiak, eds.),pp.443-448, Gordon and Breach, 1972.

16. Kutsuzawa, J., Saida, M. and Ando, T.
Colour graphic display with surface painting, Fujitsu Scientific and Technical Journal, Vol.11, No.2, pp.1-14, June 1975.

17. Grimsdale, R.L., Willis, P.J. and Hadjiaslanis, A.A.
The zone management processor: a module for generating surfaces in raster-scan colour displays, IEE Journal on Computers and Digital Techniques, Vol.2, No.1, pp.21-25.

18. Spencer, D.R. and Huang, T.
Bit plane encoding of continuous tone pictures, Proceedings of the Symposium on Computer Processing in Communications, Polytechnic Institute of Brooklyn, New York, pp.101-120, April 1969.

19. Huffman, D.A.
A method for the construction of minimum-redundancy codes, Proceedings of the IRE, Vol.40, No.9, pp.1098-1101, September 1952.

20. Schreiber, W.F.
Picture coding, Proceedings of the IEEE, Vol.55, No.3, pp.320-330, March 1967.

21. Edmonds, E.A., Schappo, A. and Scrivener, S.A.R.
Image handling in two-dimensional design, IEEE Computer Graphics and Applications, pp.75-88, July 1982.

22. Knowlton, K.
Progressive transmission of grey-scale and binary pictures by simple, efficient and lossless encoding schemes, Proceedings of the IEEE, Vol.68, No.7, pp.885-896, July 1980.

23. Warnock, J.E.
A hidden-surface algorithm for computer generated halftone pictures, University of Utah Computer Science Department, Report TR 4-15, 1979.

24. Tanimoto, S. and Pavlidis, T.
A heirarchical data structure for picture processing, Computer Graphics and Image Processing, Vol.4, pp.104-119, 1975.

25. Horowitz, S.L. and Pavlidis, T.
Picture segmentation by a tree traversal algorithm, Journal of the ACM, Vol.23, No.2, pp.368-388, April 1976.

26. Klinger, A. and Dyer, C.R.
Experiments on picture representation using regular decomposition, Computer Graphics and Image Processing, Vol.5, pp.68-105, 1976.

27. Klinger, A. and Rhodes, M.L.
Organisation and access of image data by areas, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.PAMI-1, No.1, pp.50-60, January 1979.

28. Hunter, G.M. and Steiglitz, K.
Operations on images using quad trees, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1, No.2, pp.145-153, April 1979.

29. Hunter, G.M. and Steiglitz, K.
Linear transformations of pictures represented by quadtrees, Computer Graphics and Image Processing, Vol.10, No.3, pp.289-296, July 1979.

30. Samet, H.
Region representation: quadtree-to-raster conversion, University of Maryland Computer Science Center Report TR-768, June 1979.

31. Samet, H.
Region representation: quadtrees from boundary codes, Communications of the ACM, Vol.23, No.3, pp.163-170, March 1980.

32. Dyer, C.R., Rosenfeld, A. and Samet, H.
Region representation: boundary codes from quadtrees, Communications of the ACM, Vol.23, No.3, pp.171-179, March 1980.

33. Samet, H.
Connected component labeling using quadtrees, Journal of the ACM, Vol.28, No.3, pp.487-501, July 1981.

34. Kawaguchi, E. and Endo, T.
On a method of binary-picture representation and its application to data compression, IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-2, No.1, pp.27-35, January 1980.

35. Woodwark, J.R. and Quinlan, K.M.
The derivation of graphics from volume models by recursive division of the object space, Proceedings of CG 80 Conference, 1980.

36. Woodwark, J.R. and Quinlan, K.M.
Reducing the effect of complexity on volume model evaluation, CAD Journal, Vol.14, No.2, pp.89-95, March 1982.

37. Woodwark, J.R.
The explicit quad tree as a structure for computer graphics, The Computer Journal, Vol.25, No.2, pp.235-238, 1982.

38. Oliver, M.A. and Wiseman, N.E.
Operations on quadtree encoded images, The Computer Journal, Vol.26, No.1, pp.83-91, 1983.

39. Woodwark, J.R.
Compressed quad trees (to appear in The Computer Journal).

40. Gargantini, I.
An effective way to represent quadtrees, Communications of the ACM, Vol.25, No.12, pp.905-910, December 1982.

41. Oliver, M.A. and Wiseman, N.E.
Operations on quadtree leaves and related image areas, The Computer Journal, Vol.26, No.4, pp.375-380, November 1983.

42. King, T. and Knight, B.
Programming the M68000, pp.107-138, Addison-Wesley, 1983.

43. Santalo, L.A.
Integral Geometry and Geometric Probability, Addison-Wesley, 1976.

44. Revellat, J. and Schmitt, F.
A model for colour vision, Proceedings of the 11th Congress of the International Commission for Optics, pp.101-104, September 1978.

45. Faugeras, O.D.
Digital colour image processing within the framework of a human visual model, IEEE Transactions on Acoustics, Speech and Signal Processing, ASSP-27, No.4, pp.380-393, August 1979.

46. Young, T.
Philosophical Transactions, 92, pp.12-48, 1802

47. Pratt
Part 1, Chapter 3, Photometry and Colorimetry, pp.51-90.

48. Ostwald, W. (Birren, F. ed.)
The Colour Primer, Van Nostrand Reinhold, 1969.

49. Davies, S. and Metrick, L.
An easy language for talking with colour machines, Tekscope, Vol.10, No.4, pp.7-9.

50. Straayer, D.H.
Hoisting the colour standard, Computer Design, Vol.21, No.7, July 1982

51. Willis, P.J.
The use of colour information in raster scan display systems, Displays, Vol.1, No.1, pp.47-48, April 1979.

52. Buchanan, M.D. and Pendergrass, R.
Digital image processing: can intensity, hue and saturation replace red, green and blue?, Electro-Optical Systems Design, pp.29-36, March 1980.

53. Meyer, G.W. and Greenberg, D.P.
Perceptual color spaces for computer graphics, ACM Computer Graphics, Vol.14, No.3, pp.254-261, July 1980.

54. Laycock, J. and Viveash, J.P.
Calculating the perceptibility of monochrome and colour displays viewed under various illumination conditions, Displays, Vol.3, No.2, pp.89-99, April 1982.

55. Judd, D.B. and Wyszecki, G.
Colour in Business, Science and Industry, Third Edition, Wiley, 1975.

56. Munsell Colour Company
Munsell Book of Colour, Neighbouring Hues Edition, 1970.

## Appendix 1 LEAFCODE - SCREEN POSITION TRANSFORMATION

A leaf of a quadtree encoded picture may be conveniently represented by a string of quarternary digits describing the route taken through the tree (3.2.4). For example, in a picture resolved to 512 x 512 pixels a nine digit string is sufficient to specify an individual pixel. By associating the codes 0,1,2,3 with the quadrants NW, NE, SW, SE respectively it is a simple matter to translate the successive quarterings into separate binary divisions of the horizontal and vertical extent. If the quarternary digits are each expressed as two bits then the least significant bit is associated with the horizontal (x) position and the most significant bit with the vertical (y) position. Thus, the pixel with a leafcode:

132021132 or (01)(11)(10)(00)(10)(01)(01)(11)(10)

has coordinates:

x = 110001110 binary = 398 decimal

y = 011010011 binary = 211 decimal

with the origin at the top left hand corner of the picture. Conversely, the route through a quadtree can be obtained by merging the screen (x,y) coordinates.

Appendix 2 CODING FOR COLOUR

The use of colour in computer graphics has become common in recent years. It is probably true to say that this development was mainly product-led with graphics users tempted by decreasing cost to experiment with colour in applications where it would previously have been thought inappropriate. The result is that colour, in many instances, is used as a discriminator to enhance the readability of displayed information and one is concerned mainly with the psycho-visual effects of various colour combinations. Used in this way, colour is a parameter which presents no particular coding problems; the colour range required is generally small enough to allocate different codes to different colours in a quite arbitrary way. In contrast, the display of pictures is likely to employ a much wider range of colours and this invites the use of a coding scheme which treats colour in a more systematic manner. An elaborate and extensive colour capabilty can only be of general use if it relates at some point to the way in which humans organise their perception of colour.

1. Colour perception

When called upon to describe a particular colour an observer is likely to use words like "reddish" or perhaps "bluey-green". This attribute of colour, known as hue, is clearly not sufficient for a full specification. Qualifying words such as "dark" or "bright" might be used and these refer to the relative luminance (or intensity) of the light source. A further attribute, known as

saturation, is used to describe the purity of the colour, or more exactly, the extent to which it differs from a grey colour with the same luminance. For example, a particular pink colour may be thought of as a low saturation red. These colour attributes are most easily represented on a diagram (fig. 1a) using cylindrical polar coordinates. The axis of the cylinder represents all the grey shades from black at the bottom to white at the top. On the surface of the cylinder are distributed all the "pure" colours.

Modern theories of colour vision [44,45] are based on a tristimulus model first postulated by Thomas Young in 1802 [46]. This model assumes that the eye contains three distinct types of photosensitive receptor, each with a frequency response over a limited section of the visible spectrum. Recent experimental evidence suggests that signals from these three receptors are combined to yield three signals A, C1 and C2. The achromatic response A is the only one of the three which alters when the intensity of a coloured light is varied. The chromaticity signals C1 and C2 are related to our perception of hue and saturation. By using A, C1 and C2 as coordinates it would therefore be possible (in principle) to define a colour space, similar to the one already described, which corresponds closely to the way in which colour is perceived (fig. 1b).

The intuitive colour solid described above, while serving as a useful model, lacks any potential for quantitative measurement. The development of colorimetry is described in [47] and some of the most relevant aspects are summarised below. Application to the

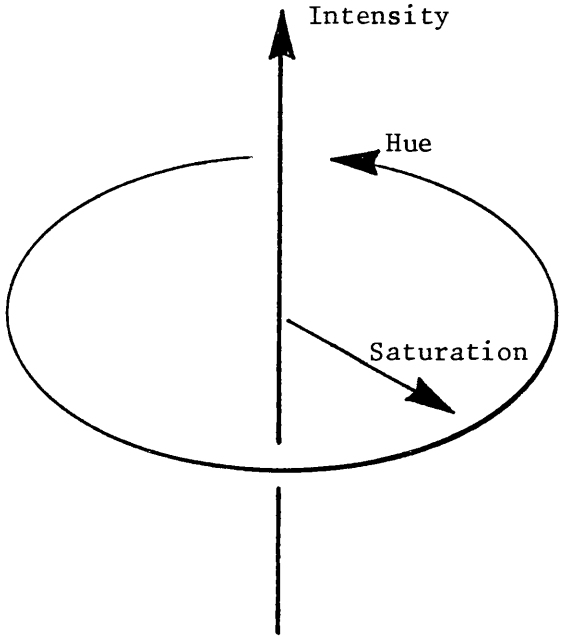



Figure 1a IHS cylindrical coordinates ....

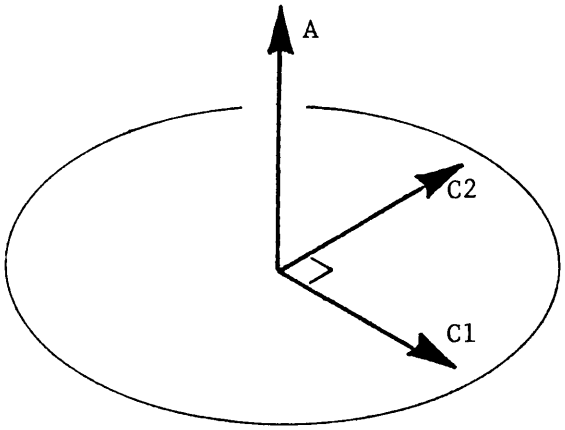



Figure 1b .... and the A, C1, C2 space

problem of CRT display is discussed in the sections which follow.

## 2. Colorimetry

The tristimulus theory is confirmed by colour matching experiments which show that any colour can be matched by a mixture of no more than three coloured lights (primaries). This may be written as

$$C = a(P1) + b(P2) + c(P3)$$

where a,b,c are the quantities of each primary required to obtain a match with colour C. The range of matchable colours is, however, limited if only positive quantities of each primary are allowed. The significance of a negative value in the colour match may be understood from the following example. Suppose that a quantity of primary P1 needs to be added to colour C in order to give a match with a mixture of the remaining primaries P2 and P3,

ie. $C + f(P1) = g(P2) + h(P3)$ where $f,g,h > 0$

then $C = -f(P1) + g(P2) + h(P3)$

This problem may be circumvented by defining a set of artificial primaries which allow a linear transformation of the set of triplets (a,b,c) to a set (a',b',c') which are positive for all colour matches.

In 1931 the Commission Internationale de l'Eclairage (CIE) adopted a coordinate system for colorimetry based upon three artificial primaries X, Y and Z. Thus, any colour can be represented by the equation

$$C = a(X) + b(Y) + c(Z) \quad \text{where} \quad a,b,c > 0$$

or more simply

$$C = X + Y + Z$$

Normalised coordinates are defined by

$$x = \frac{X}{X + Y + Z}$$

$$y = \frac{Y}{X + Y + Z}$$

$$z = 1 - x - y$$

Figure 2 is a chromaticity diagram based on the CIE X-Y-Z primaries. The full range of natural colours is bounded by a locus which represents the monochromatic spectral colours. The shaded triangle represents the range of colours which may be obtained by additive mixing of the real primaries R (red), G (green) and B (blue) represented by the vertices of the triangle. If the x,y coordinates of the R,G,B primaries are known then a linear transformation may be applied to translate any x,y into the coordinate system of the real primaries.

## 3. R-G-B colour mixing

The colour image produced by a CRT is generated by the excitation of a matrix of phosphor dots (or strips) deposited on the screen. Three suitably chosen phosphor types allow the display of a wide colour range according to the colour mixing principles outlined above. Hence, the output stages of any colour display system are inevitably R-G-B orientated; indeed the specification of colour throughout the majority of display systems is in terms of

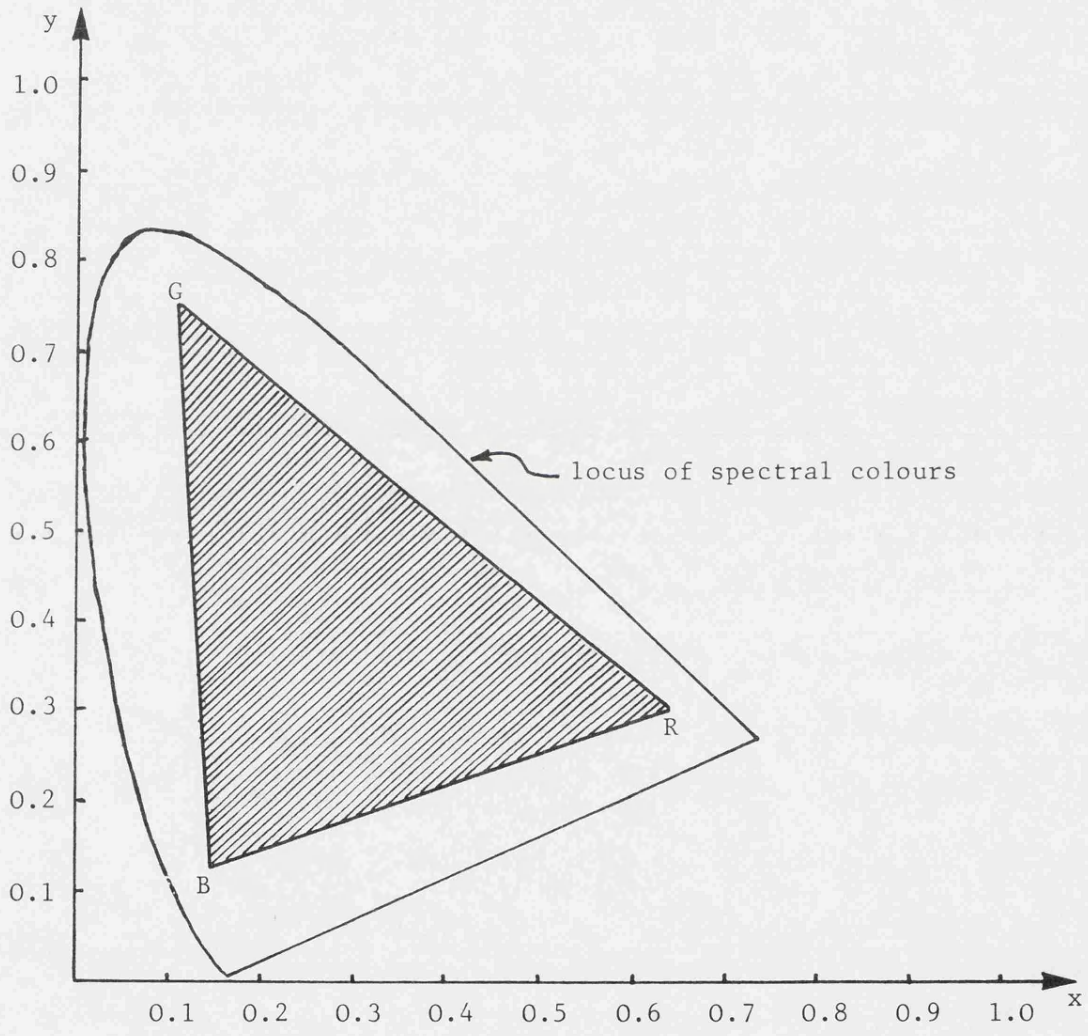



Figure 2 CIE (1931) chromaticity diagram

R-G-B mixes. Let us consider the consequences imposed by such an organisation assuming that the monitor is set to display white for equal R, G, B signal inputs. The most elementary colour code requires one bit for each primary and could display eight colours:

| R | G | B | |
|---|---|---|---|
| 0 | 0 | 0 | black |
| 1 | 0 | 0 | red |
| 0 | 1 | 0 | green |
| 0 | 0 | 1 | blue |
| 1 | 1 | 0 | yellow |
| 1 | 0 | 1 | magenta |
| 0 | 1 | 1 | cyan |
| 1 | 1 | 1 | white |

Figure 3 shows how these colours would be placed on an intensity, hue, saturation (IHS) diagram. It has been assumed that the single blue, green and red primary colours are ranked equally luminous although this is unlikely to be true. If the number of bits per primary is increased the colour space becomes more densely populated and occupies a volume which approximates to a double cone structure (fig. 4). There will be no saturated colours near the top of the solid because the brightest colours must have all non-zero R, G and B components ie. a grey component. A double cone model of this type, named the Ostwald Colour Solid [48], has recently been revived by Tektronix [49,50] as the basis of a colour standard for their graphics systems. Their vertical axis is named Lightness and the other variables are Hue and Saturation. It is important to understand the relationship (illustrated in figure 5) between this HLS model and the cylindrical IHS model. It seems to this author that the lower conical shape is the result of a (quite reasonable) distortion of the cylinder which reduces the black point (I=L=0) to

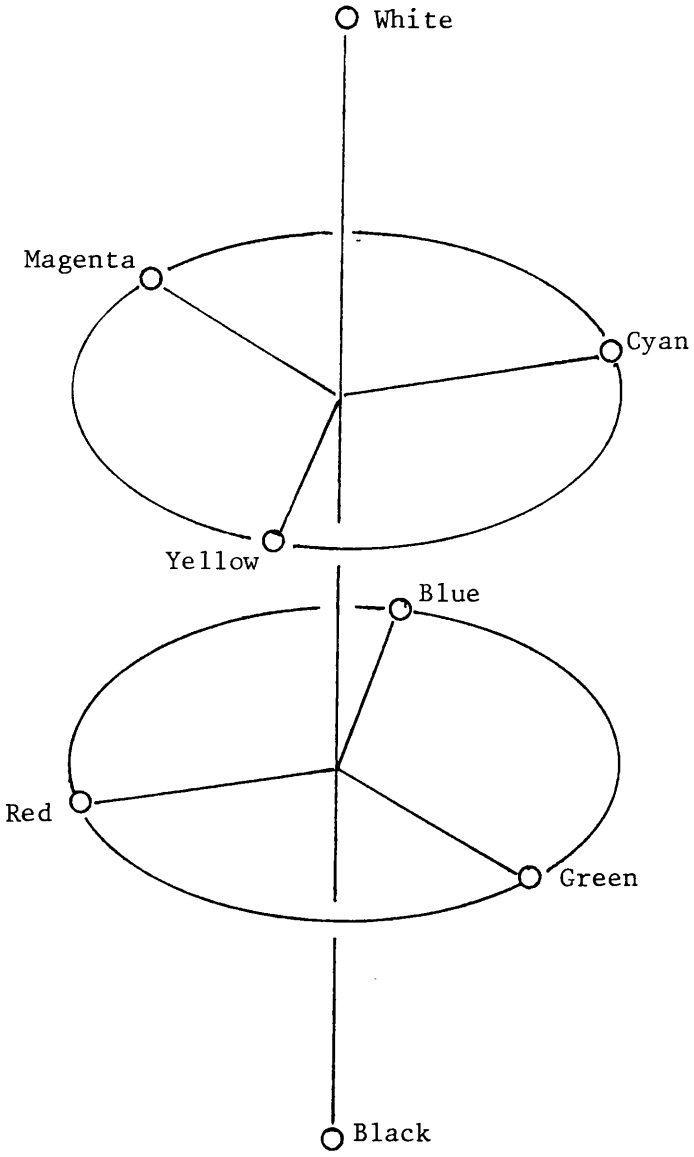



Figure 3 Eight colours in the IHS space

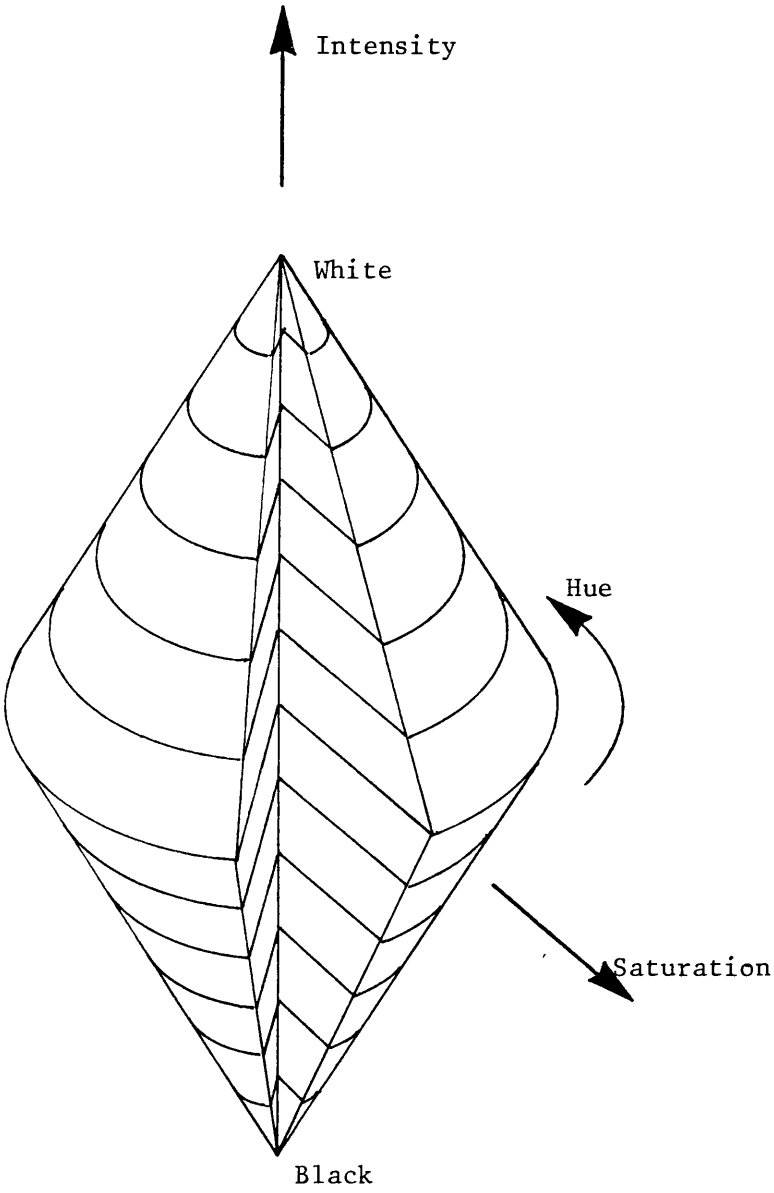



Figure 4 The double cone colour model

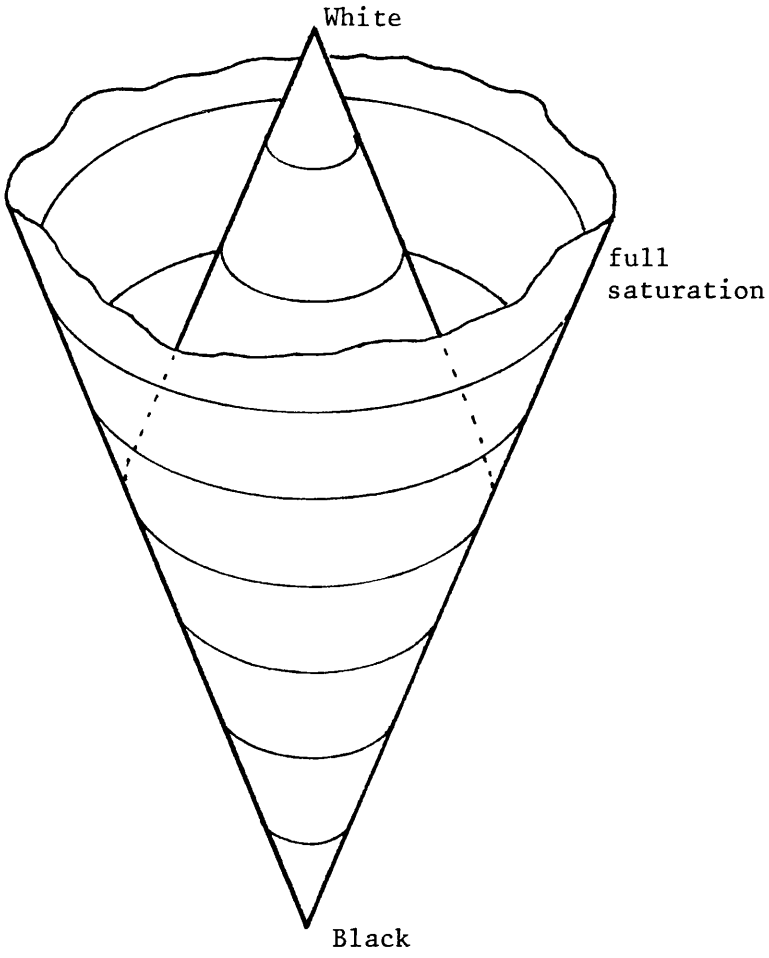



Figure 5 The double cone related to IHS

a singularity. All the points on the surface of this theoretically infinite cone represent fully saturated colours. The significance of the upper cone may be appreciated from two viewpoints. In the context of R-G-B colour mixing it is simply the result of each primary having an upper limit. This is the only reason why it is not possible to represent red (for example) at the same intensity as brightest white. Alternatively, the double cone may be regarded as a representation of the range of observable colours when a non-luminous object is illuminated by a white reference source. The white point is the result of 100% relection of the incident light: all other colours are the result of components of the incident light being absorbed. In either case it seems misconceived to refer (as does Tektronix) to all colours on the surface of the upper cone as having 100% saturation.

## 4. The organisation of colour for computer graphics

In the previous section it was shown how a 3 bit colour code could be used to generate a range of eight colours by associating one bit with each of the RGB outputs. A greater flexibility of coding can be obtained by the inclusion of a hardware colour look-up table (CLT) which maps each of the colour codes to a prescribed mixture of R-G-B components (fig. 6). The number of colours in the palette remains limited by the bit length of the colour code but the specification of each colour in the palette becomes a function of the number of bits allocated to each of the R-G-B primary outputs. One of the many advantages which a CLT offers is the ability to alter values in the map and thus remix the colours

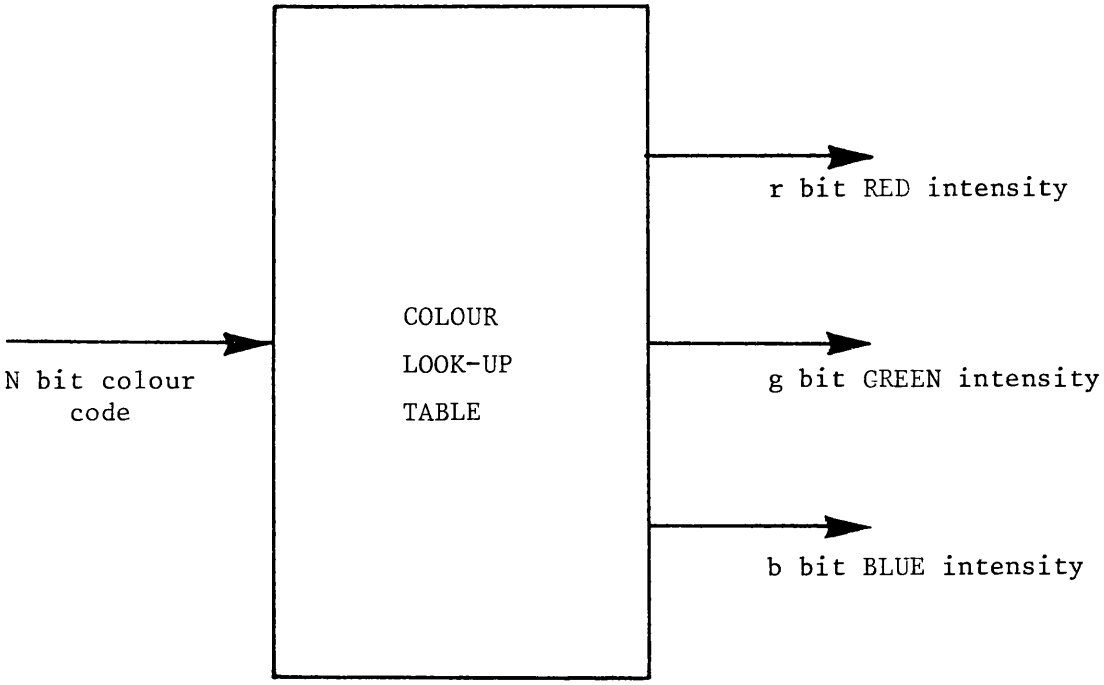



Figure 6 Colour mapping

available to the user.

From the graphics user's point of view the R-G-B system is unnatural and consequently difficult to manipulate. This problem has been discussed by several authors [51-53] who advocate a colour model more closely aligned to the perceptual parameters of intensity, hue and saturation. This suggests an application of the CLT as a translator from IHS to RGB in which the colour code would be divided into separate I,H and S fields. Buchanan and Pendergrass [52] have suggested that 6 bits for intensity, 5 bits for hue and 3 bits for saturation are sufficient for TV monitor display. However, their colour plates reveal that this degree of resolution is not adequate to produce visually smooth colour transitions. The most compelling argument against the use of such a code is its extreme inefficiency. This is a consequence of the geometry of the perceptual colour space which (in line with common sense) predicts that more HS combinations are required as intensity increases and that discrimination between hues must be improved as saturation increases. The use of fixed width IHS fields to represent non-independant parameters is clearly wasteful and an alternative method must be sought for ordering the colour code. This will be dealt with in section 7.

Assuming the colour code can be sensibly ordered it must next be considered how best to fill the CLT with RGB component values.

## 5. Uniform colour spaces

The organisation of colours into a perceptual space resembling a double cone shape is an established practice which, as has been

shown, corresponds conveniently to the generation of colour by RGB colour mixing. Given a system with the ability to display a fixed number of colours a method is required which distributes these colours evenly throughout perceptual colour space. This demands a quantitative colour space within which just noticeably different colours are equally spaced.

Investigation into the ability of subjects to detect small colour differences reveals a substantial perceptual non-linearity in the CIE (1931) chromaticity diagram [54]. The shaded areas in figure 7 each contain approximately the same number of discernable colours and illustrate the fact that individuals discriminate poorly in the red part of the diagram compared with the blue and least effectively in the green.

A number of systems have been proposed for transforming the CIE (1931) diagram to provide a uniform chromaticity scale (UCS). The simplest of these is the CIE (1976) UCS which is the result of a linear transformation of the X-Y-Z coordinates according to the formulae:

$$u' = \frac{4X}{X + 15Y + 3Z} \quad \ldots\ldots (1)$$

$$v' = \frac{9Y}{X + 15Y + 3Z} \quad \ldots\ldots (2)$$

When experimental colour difference thresholds are replotted using these coordinates the linearity is significantly improved [54]. The CIE (1976) L*-u*-v* system is a formulation of a three dimensional colour space based upon the u',v' chromaticity scale

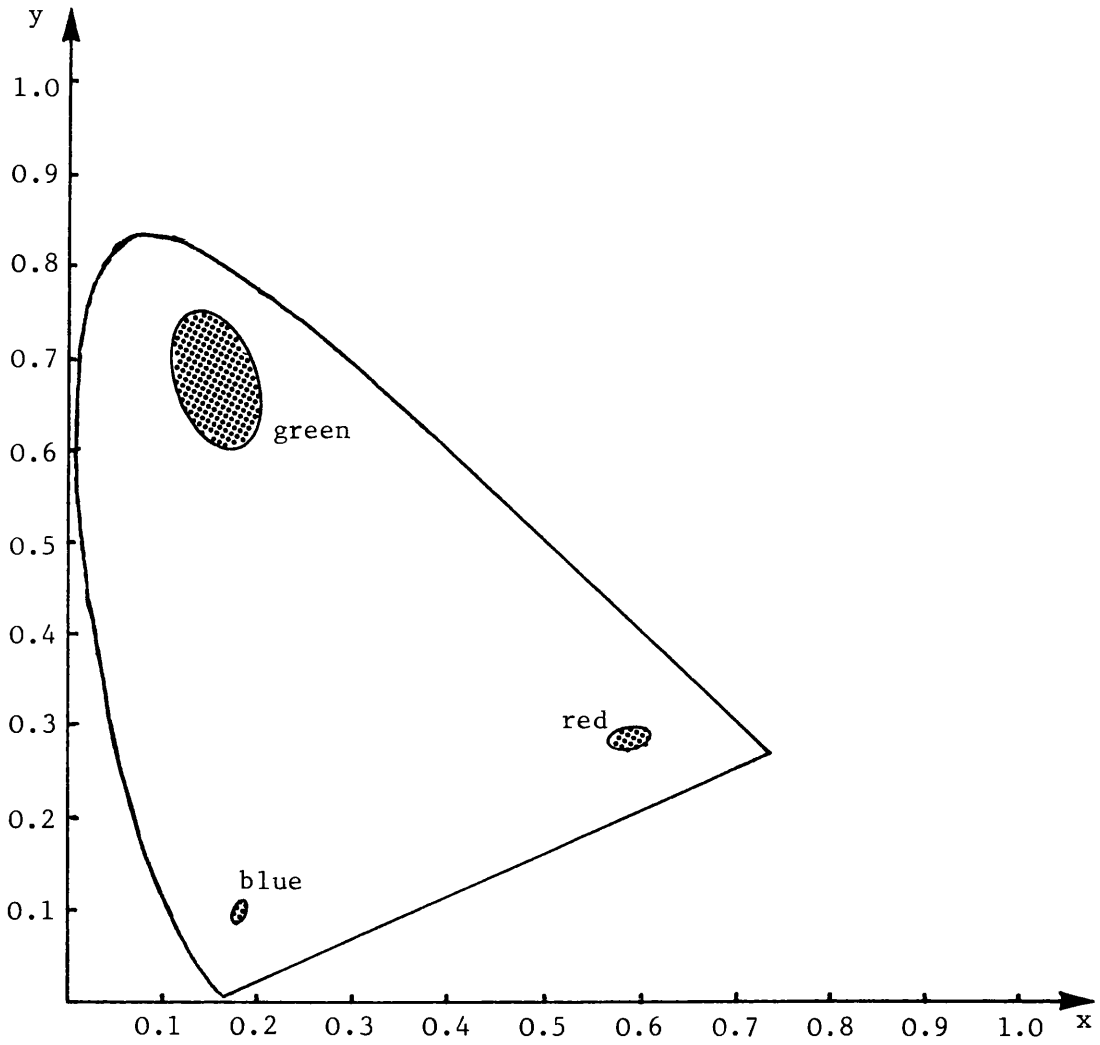



Figure 7 Perceptual non-linearity in the CIE (1931) chromaticity diagram

with

$$L^* = 25\,Y^{1/3} - 16 \quad \ldots\ldots\ldots (3)$$

$$u^* = 13L^*(u'-u_0') \quad \ldots\ldots\ldots (4)$$

$$v^* = 13L^*(v'-v_0') \quad \ldots\ldots\ldots (5)$$

where $X_0,Y_0,Z_0$ and $u_0',v_0'$ are the coordinates of reference white normalised so that $Y_0 = 100$. A diagram showing the form of the L*-u*-v* space is given in [55] p.331. The irregular shape resembles the empirically derived Munsell colour solid [56].

## 6. Derivation of an evenly graded colour look-up table

Assuming that the CIE (1976) L*-u*-v* coordinate system provides a good approximation to a uniform colour space it is reasonably simple to calculate the R,G,B coordinates of evenly distributed colours. Equations (1)-(5) above may be rewritten to express X,Y,Z in terms of L*,u*,v* as follows:

from (4) $$u' = \frac{u^*}{13L^*} + u_0' \quad \ldots\ldots\ldots (6)$$

from (5) $$v' = \frac{v^*}{13L^*} + v_0' \quad \ldots\ldots\ldots (7)$$

from (3) $$Y = \left(\frac{L^* + 16}{25}\right)^3 \quad \ldots\ldots\ldots (8)$$

Solving (1) and (2) gives

$$X = \frac{9u'Y}{4v'} \quad \ldots\ldots\ldots (9)$$

$$Z = \frac{(12 - 20v' - 3u')Y}{4v'} \quad \ldots\ldots (10)$$

The relationship between the CIE (1931) X-Y-Z coordinates and the R,G,B coordinates depends upon the exact specification of the display phosphors. Thus

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} kx_r & lx_g & mx_b \\ ky_r & ly_g & my_b \\ kz_r & lz_g & mz_b \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad .... (11)$$

so that for $\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ we find $\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = k \begin{pmatrix} x_r \\ y_r \\ z_r \end{pmatrix}$ etc.

The constants k,l,m may be determined by the condition

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \text{for reference white} \quad \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix}$$

which means solving

$$\begin{pmatrix} x_r & x_g & x_b \\ y_r & y_g & y_b \\ z_r & z_g & z_b \end{pmatrix} \begin{pmatrix} k \\ l \\ m \end{pmatrix} = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} \quad ..... (12)$$

Equation (11) may now be written

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

and the inverse matrix M can be calculated which provides the transformation

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = M \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

There is one more factor which must be considered before the CLT may be assigned RGB values and this relates to the non-linearity of the CRT display. The intensity of light P generated by

a CRT primary phosophor is related to the input signal voltage Vp by the formula

$$P = k\ Vp^{\gamma}$$

Signal voltages are proportional to the values stored in the CLT so these values (r,g,b) may be calculated from

$$r = R^{1/\gamma} \times n \qquad g = G^{1/\gamma} \times n \qquad b = B^{1/\gamma} \times n$$

where n is the value corresponding to R or G or B = 1. For example, if each primary is represented by an 8 bit value then n=255.

A preliminary trial has been conducted by the author using the method described above. A set of 1850 r,g,b triplets were obtained by translating values based on a cubic grid of spacing 10 units in L*-u*-v* space. These fall within an L*-u*-v* subspace limited by the represention of the primaries red, green, blue and the secondaries yellow, magenta, cyan as shown in figure 8. When colours of constant L* are displayed on the monitor the transitions between adjacent u*,v* colours are almost imperceptible. However, when L* is varied for constant u*,v* the transitions are very noticeable; a not surprising result since the grid allows only eleven intensity levels from L* = 0 to L* = 100. Further investigation is planned and it appears that the constant 13 which appears in equations (4) and (5) may need to be reduced to provide equal perceptual spacing in the L* direction and the u*,v* plane.

## 7. Using the colour table

Following the previous discussion it is proposed that the CLT should contain r,g,b values associated with each of the colour codes and that they should be ordered according to their L*,u*,v*

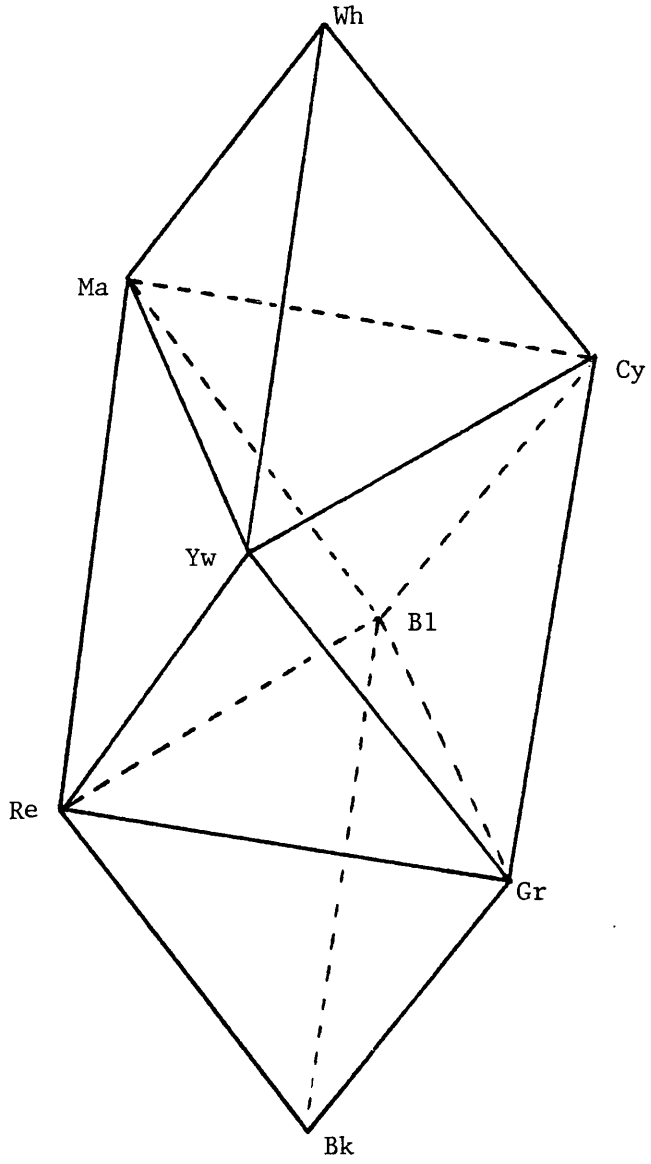



Figure 8 The realisable L*-u*-v* subspace

designation. An auxilliary table must exist which makes these L*,u*,v* values explicit.

| Colour code | Active CLT | | | Auxilliary table | | |
|---|---|---|---|---|---|---|
| 0 | r0 | g0 | b0 | L*0 | u*0 | v*0 |
| 1 | r1 | g1 | b1 | L*1 | u*1 | v*1 |
| ... | .. | .. | .. | ... | ... | ... |
| n | rn | gn | bn | L*n | u*n | v*n |

The auxilliary table allows sensible colour calculations to be performed. For example, the average of two colours may be found by averaging their L*,u*,v* components. If the result is rounded to the nearest interval then the colour code of the average can be located by searching the auxilliary table for an L*,u*,v* match. Colours may be modified interactively using intensity, hue and saturation as parameters. For example, to locate colours with the same intensity and hue as a given colour it is merely necessary to calculate the ratio k = u*/v* and then, depending on the value k, select from the table the set of colours kv*,v* or u*,u*/k.

Appendix 3 QUAD ENCODED DISPLAY

This paper has been accepted for publication in

IEE Proceedings-E, Computers and Digital Techniques

# Quad encoded display

D.J. Milford, B.Sc., A.R.C.S., G.Cert.Ed., and P.J. Willis, B.Sc., M.Sc., D.Phil., C.Eng., M.I.E.E., M.B.C.S.

*Indexing terms:* Image processing, image processing & pattern recognition, Computer applications

**Abstract:** A colour raster display system based on picture encoding is described. Encoding pictures as a means of data compression has a long history and is a technique of interest to anyone working with complex pictures. Its primary value is the lower bandwidths required of transmission channels, but its more frugal use of store is not to be overlooked. A quad tree as a means of encoding area-coherent pictures is used because it retains spatial information in a form which is still amenable to processing. The combination of this with compression makes this a particularly useful form of coding for raster-style colour pictures, and this paper addresses the problems which had to be overcome in order to construct a display directly exploiting the quad tree and related forms. The potential of such a display for rapidly handling very large amounts of pictorial data and for application to picture-archive retrieval is also discussed. The system has been constructed, is fully operational and incorporates roam and zoom operations. Sample pictures are included.

## 1 Introduction

A traditional view of pictorial data compression is that it is a helpful means of making full use of limited capacity transmission channels. It is necessary to express the picture in its full form once it has been received, in particular before it can receive any further processing. In this sense compression is only an intermediate, although very useful, process which can largely be treated as a pair of black boxes. One box sits at the source end of the transmission channel and performs encoding, and the second box is attached at the receiving end and decodes. The use of this idealised system is not concerned with the transmission format or the nature of the black boxes. This model applies to data compression in general and not just to the transmission of pictures. However image-like pictures are very demanding of bandwidth and storage, more so than most transmitted data, and so there is special interest in this [1–3]. By image-like, we mean framestore pictures which are created at the pixel level, as distinct from calligraphic; i.e. those which are readily expressed as a series of plotter-like movements of a hypothetical pen loaded with coloured ink. Framestores may certainly be used for calligraphic applications, and the command sequence needed to realise the picture would normally be a very compact representation in itself. However, the image is a more generally useful form because it represents the picture directly, pixel by pixel, and so is readily shared with other users without presuming some application-specific underlying data structure [4, 5]. Unfortunately it is also the most demanding in terms of data: a typical framestore for image-like applications will require at least 256 kbytes of storage for each picture. If the image has been captured, rather than synthesised, this requirement may be tripled, corresponding to 512 × 512 resolution with 24-bit direct colour. The prospect of 1024 × 1024 displays becoming standard in the near future pushes this requirement four times further to 3 Mbytes for every image.

Image manipulation is also a task which is greatly impeded by the large quantities of data. Indeed it is only in the last two years or so that it has been possible to buy microprocessors with a large enough memory space to be able to address complete pictures of the sizes just mentioned, without recourse to overlays or other inconvenient techniques. Image processing (in the conventional sense of feature extraction etc.) has thus tended to be a mainframe task and has certainly been batch oriented except where expensive, specialised hardware has been available. Image manipulation (in the wider sense as a means of interacting with pictures) has not been feasible with other than very low resolution representations, and consequently users of images have been well advised not to use computing resources shared with others. The rapidity with which images can fill a disc or overload a network is a well known hazard, while the interchange of pictorial information by readily available inexpensive serial interfaces is a very tedious process, and prone to mishap.

Clearly, encoding overcomes the major problem of the amount of data, although it must always be recognised that coding may impose limitations on the nature of pictures which can satisfactorily be compressed. What is required for transmission and storage is a technique which gives good compression. What is required when manipulating such pictures can also, with great advantage, be a compressed form, provided also that the structure of the image is maintained explicitly. By 'the structure of the image' we do not mean an underlying data structure such as might be used to represent, for example, a technical drawing. Rather, we mean the physical structure as seen on the display: adjacent areas must still be identifiable, as should regions of constant colour. In this way the encoding is in a form which the user can understand directly by reviewing the picture. We suggest that, while there may be several ways of implementing it, this approach is the only satisfactorily general way of coding images. It is not being suggested that other approaches to coding are invalid. On the contrary they are frequently the most useful form. However, we are claiming that such techniques are application specific or are largely meaningless to the viewer. The most general approach to image manipulation is clearly to use the pixel form, where every item corresponds directly to what is seen. Our approach is to maintain this direct correspondence with what is on screen, rather than with what it represents, while introducing compression. It has proved possible to construct a display based on this approach. The encoding method will now be introduced, then the display will be described in detail.

## 2 Representation of pictures

### 2.1 Quad tree

To identify a form of compression to meet the requirements implied in the previous paragraph, it seemed reasonable to study the image-processing literature. A technique which has found acceptance is the use of a quad tree [6–11]. Under this scheme the picture area is divided

Paper 1136E (C2, C5) first received 22nd August 1983 and in revised form 29th February 1984

The authors are with the School of Mathematics, University of Bath, Claverton Down, Bath BA2 7AY, England

recursively into quarters until the areas are either uniform in colour or until the subdivided area covers only one pixel (Fig. 1). The picture is thereby represented as a four-branched tree in which the leaves correspond to square

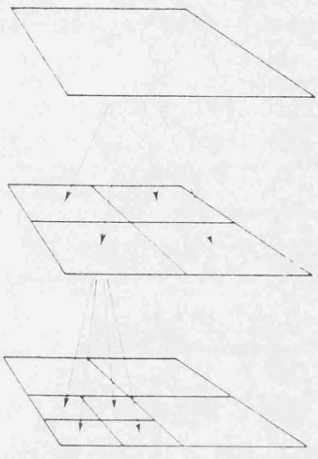


Fig. 1 *Recursive division of the picture area into quads*

areas of uniform colour, termed 'quads' (Fig. 2A). A quad may be encoded by its size (always a power of two) and its colour. The tree thus exploits area colour coherence to represent the picture, while maintaining spatial information in the branches. As will be seen, this representation also results in data compression. Indeed, while it may be defeated (as may other coding schemes) by constructing pathological pictures in which every pixel differs from its neighbours, the fact that it has already found independent use in representing digitised images is certainly a recommendation [8].

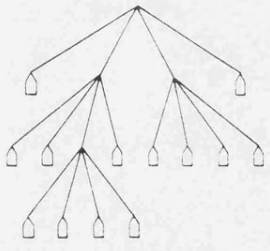


Fig. 2A *Quad-tree representation*

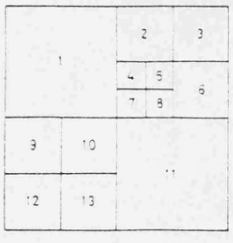



Fig. 2B *Corresponding picture with its quads labelled in scan order*

### 2.2 Quad list

The quad tree is a most useful representation, but a related form is also of value and is central to the working of the display system described shortly. This form is the quad list. In essence it is a list of quads in scan order. Scan order means the sequence in which the quads are first met by a conventional raster scan; this order is indicated by the numbering in Fig. 2B. The quad list thus very closely defines the on-screen picture, but spatial information is now implicit rather than explicit.

## 3 Quad-list display processor

The display system to be described uses the technique of quad encoding to achieve data compression, enabling efficient transmission and storage of high resolution colour pictures. A picture is decoded from a quad list by dedicated real-time hardware avoiding the necessity for a full bit-mapped display memory. We will later describe how, at no extra hardware cost, the display may be used to roam over large pictures and to zoom in to reveal finer detail.

### 3.1 Picture format

The present system displays pictures at 512 × 512 spatial resolution, with, for any single picture, upto 4096 colours chosen from a palette of over 16 million. Pictures are transmitted and stored in the form of a quad list, each element of which describes the colour and size of one quad. The colour is held as a 12-bit code and, because the quad size is always a power of two, the size may be held as a 3-bit code (Table 1). Differentiation of odd and even field

**Table 1 Quad codes**

| Quad dimensions | Size | Height | Width |
|---|---|---|---|
| Even field pixel | 0 | 0 | 0 |
| Odd field pixel | 1 | 0 | 0 |
| 2 × 2 pixels | 2 | 0 | 1 |
| 4 × 4 pixels | 3 | 1 | 3 |
| 8 × 8 pixels | 4 | 3 | 7 |
| 16 × 16 pixels | 5 | 7 | 15 |
| 32 × 32 pixels | 6 | 15 | 31 |
| 64 × 64 pixels | 7 | 31 | 63 |

pixels eases the problem posed by raster interlace for the decoding hardware. The upper limit of 64 × 64 placed on quad size imposes negligible loss of efficiency in coding since the occurrence of larger quads is rare and their representation as several smaller ones adds very little to the length of the display list. Ordering of quads within the display list is defined by the order in which each quad is first encountered in the conventional top left to bottom right raster scan (Fig. 2B). There is one exception to this scan ordering which, again, is imposed to overcome interlace problems. Pixel-size quads are displayed only on alternate raster fields so, for reasons which will be given later, these quads are placed in the display list as even-field odd-field pairs (Fig. 3).

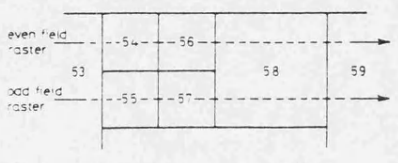



Fig. 3 *Order of pixel size quads in the quad list*

### 3.2 Overview of the display implementation

The display (Fig. 4) is managed by a Motorola 68000 16-bit processor which runs a modest operating system

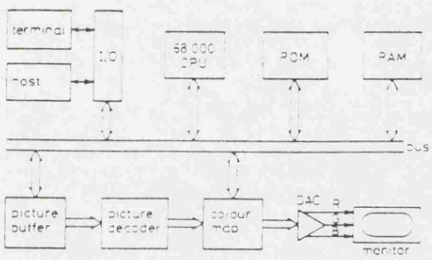



Fig. 4 *Display system*

stored in ROM. Display lists are loaded from a host machine into RAM whose present capacity of 256 kbytes permits local storage of (typically) ten pictures. Once loaded, a picture may be displayed virtually instantaneously by moving the relevant display list to the picture buffer where it is accessed by the decoding hardware. The use of compressed picture data removes the necessity for a closely coupled parallel interface to the host, because pictures may be loaded sufficiently rapidly using a conventional serial line operating at 19200 baud.

The picture buffer is dedicated to screen refresh and may hold a display list of up to 32 k quads, allowing reasonably complex pictures to be presented. Every item in the display list is accessed once per field scan and decoded in real time by the subsequent hardware. To meet the speed requirements and to avoid using fast memory, quads are read in parallel, eight at a time, from the picture buffer.

### 3.3 Picture decoder

The first stage of the decoder (Fig. 5) copes with the problem of display interlace by rejecting those quads of pixel size which are not required in the current field. Exactly half of the pixel quads will have to be rejected in the complete field. Without some form of control on the distribution of these, there would be occasions when all eight of the parallel-read quads are pixels not required on the current field. Hence they would all be rejected and the parallel read would have failed to produce any quads for the display. Indeed, there might be long sequences of such parallel reads because there could be complete scan lines consisting only of pixel quads. Hence the display hardware could not be constructed to deliver picture information in real-time synchronism with the raster scan. It is for this reason that the display list holds quads of pixel size in even-field odd-field pairs, thus ensuring at least four valid data items on each parallel fetch. With this scheme the shortest time between reads of the picture buffer will be four pixel periods, i.e. four times 70 ns. The memory used has an access time of 200 ns, giving a generous margin.

*3.3.1 Sorting the input code (Fig. 6).* Parallel data from the picture buffer is applied to an eight to one channel selector. This is controlled by a simple finite state machine whose state describes the currently active channel. The size codes from each channel are tested to isolate those quads of pixel size which are not needed for the current field. A flag is set for each channel holding a valid quad. These flags, together with the current state, determine the next active channel. The look-ahead implied in this system

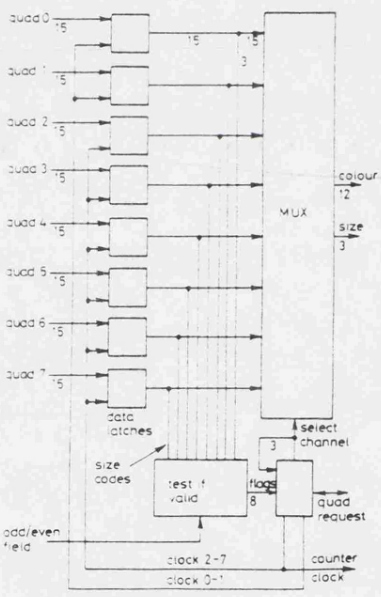



Fig. 6 *Decoder input sort*

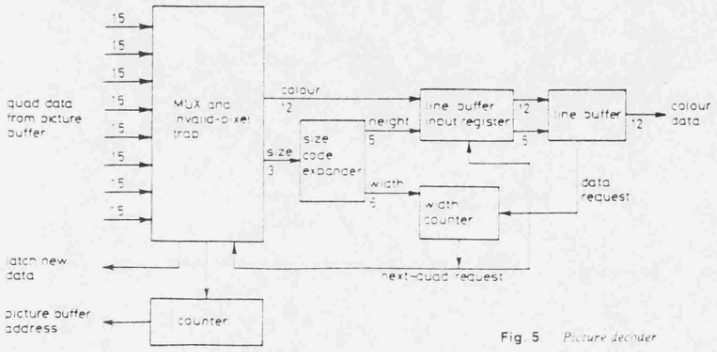



Fig. 5 *Picture decoder*

requires a prefetch on channels 0 and 1 whenever states 6 or 7 are entered.

All quads have a size which is a power of two and therefore it is the exponent rather than the actual size which is stored in the compressed form of the picture. As the quad moves downstream, the 3-bit size code thus has to be expanded (Table 1) to a width and to a height code which are used to control data entry to the line buffer in a manner which will now be explained.

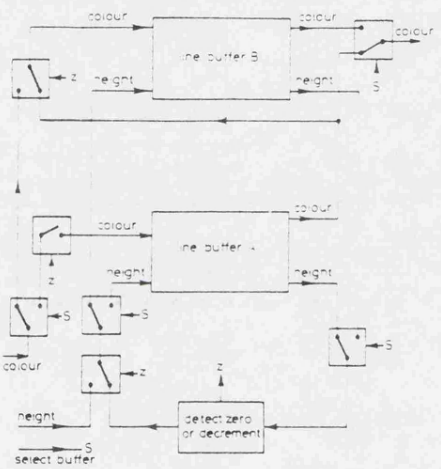



Fig. 7 *Line buffer operation*

Switch configuration
read from buffer A
write to buffer B
no zero height detected

*3.3.2 Line buffer (Fig. 7)*: The function of the line buffer is to hold information about the pixels on the current raster line. It comprises 512 locations (corresponding to the horizontal display resolution) each storing

(*a*) a 12-bit pixel colour code

(*b*) the 5-bit height code

The colour code corresponds directly to data which would occupy a pixel location in a conventional framestore. The 12-bit colour code coming from the line buffer is passed to a look-up table where it generates a 24-bit output, 8-bits each to three digital-to-analogue convertors. These produce separate red, green and blue video signals which drive the colour monitor. The look-up table is random-access memory loadable by the local processor and so a wide colour range is displayable.

The height code determines the number of scan lines for which a particular line buffer entry remains valid and is used in the following manner. During a raster line each line buffer location is examined sequentially at pixel rate, and the colour code is output. If the height code is found to be zero, a request is issued for fresh quad data to be written to the line buffer from an input register. Otherwise the colour code remains unchanged, but the height code is decremented by one and written back. For example, a quad of size 8 × 8 contains pixels on four scan lines in either field. If its height code is initially written as three, then at the fourth reading it will be zero and the relevant buffer locations will be filled with new data. The 5-bit code can therefore accommodate quads up to size 64 × 64.

With a pixel period of about 70 ns, practical considerations rule out a read decrement write cycle at a buffer location. The problem is resolved by double buffering each location, reading from one and writing to the other, with buffers switching function on alternate scan lines. The cycle time is effectively extended by delayed addressing of the write buffer.

The width code is of particular importance in filling the line buffer. Requests from the line buffer for fresh data must be counted so that the input register is updated at appropriate intervals. For example, if the current input quad is size 8 × 8 then the input register should be read eight times as requests are issued by eight consecutive empty line-buffer locations. This function is controlled by the width code which is held in a counter and decremented at each line-buffer request. If the count is zero when a request is received then, immediately after being read, the input register is reloaded with new colour and height codes, and the counter is loaded with new width code.

## 4 Performance of the display

A number of pictures have been obtained by using a volume modeller developed by colleagues in the School of Engineering [11]. These vary in complexity from 10 000 to 25 000 quads, and represent engineering components. Figs. 8–10 illustrate part of this range, with, respectively, 13 477, 13 714 and 19 516 quads being used. The pictures are coded with 12-bit colour, as described earlier, although the entire colour range is not used. Hence each quad requires two bytes of data and the times required to transmit this data down an asynchronous line at 19 200 baud

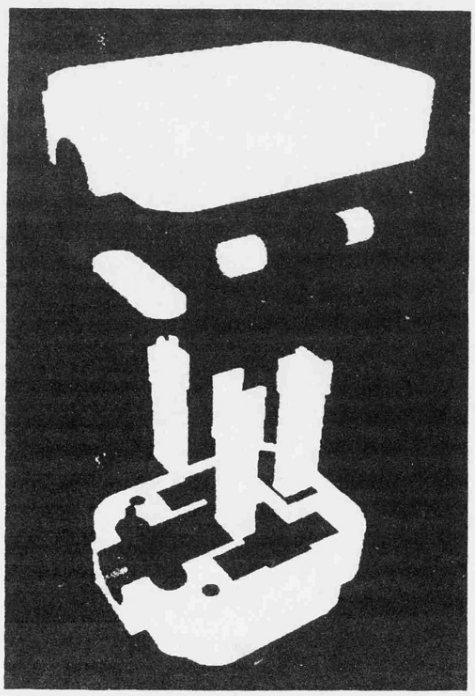


Fig. 8 *Plug*

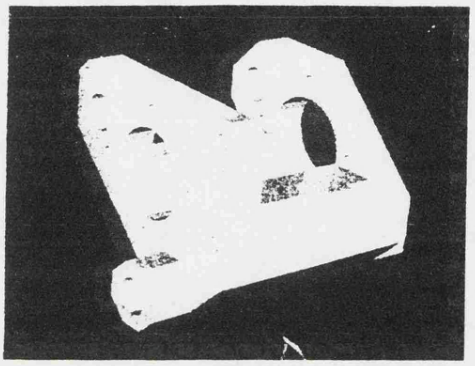


Fig. 9 *Bracket*

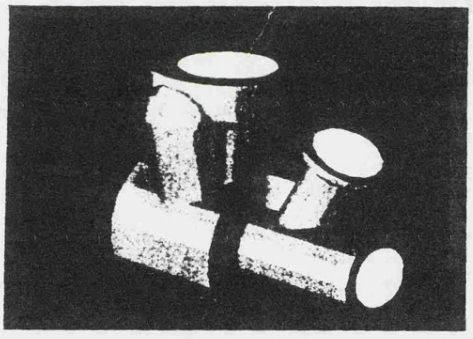


Fig. 10 *Engine*
Courtesy of Radan Computational Ltd.

are 26, 27 and 39 s. These figures include some disk latency in retrieving the pictures. There is adequate memory local to the display to hold typically ten such pictures. Any one can then be displayed immediately: the time required to replace the picture locally is approximately 0.1 s, making this system very attractive for displaying large amounts of pictorial data at high colour resolution. This type of performance greatly exceeds that of a conventional framestore which would not in any case be able to store such large amounts of pictorial data economically. Naturally there is no need to limit the interface to a low-speed serial line: we have also implemented a simple parallel interface with consequent speed-up of transmission time. A network interface to a remote filestore is also feasible without picture transmission overloading the network. All the pictures shown here are held on floppy discs attached to a simple CP/M Z80 system, adequate demonstration that sophisticated retrieval and interfacing is not needed.

The overall compression achieved in these pictures is approximately one order of magnitude, comparable to simple run-length encoding, but not as great as some variants. We attribute this to the gains yielded by representing areas being offset by the lower likelihood of the extended coherence needed by quads. However, as we have been at pains to point out, compression is not the only measure of such a system. It is also desirable to be able to manipulate the picture and this requires the tree form. We have accordingly experimented with transmitting quad trees as well as the quad lists used by the special display hardware. The results are very encouraging. First, it is possible to represent the tree with only one third more storage than needed for the list, so this is still a compact form of the picture. Secondly, using the local display processor, it is possible to compile trees into quad lists in approximately 0.5 s, this being the time for pictures of the same complexity as those shown here. Thirdly, the display software can easily prune the tree prior to compilation, allowing the viewer to be selective in what is displayed. Specifically, the viewer can roam over large pictures and zoom-in to reveal further detail. Both of these are possible because the quad tree retains explicit spatial information: roaming requires pruning the width of the tree, and zoom requires pruning the depth. In practice, the pruning is readily incorporated in the compilation process and so the tree structure is left intact. A further benefit is that no extra hardware is needed to do this.

The quantity of hardware required to implement the display is modest: the entire system only partly occupies a standard card frame of double Euro size, the cards are mainly wire wrapped and not densely occupied. TTL is used for all logical operations and cooling is entirely by natural ventilation. The cost of the display is thus that of an intelligent terminal rather than that of a framestore, ensuring that accessing a pictorial archive would not be limited by cost to users at the archive location. It is therefore sensible to think of remote access of pictures in the way we currently think of remote access to textual documents, via dial-up rather than by journeying to some highly specialised central equipment.

## 5 Future development

All our work to date points to the value of compression in output only applications. The virtues of quad trees for image processing have been extolled by others and will not be repeated here. The particular value that we put on this form of compression is that it is readily amenable to coding pictures of greater than screen size. Coupled with the capacity of our display to hold several picture frames locally, instantly viewable, it seems that it fills an obvious niche in the retrieval of pictures from a pictorial data base, especially when it is necessary to roam over large pictures at various levels of detail. This is an interesting development: quad trees were originally put to use for processing pictures and we are now proposing a return to their earliest application as a method of picture representation, but in the context of picture retrieval rather than image processing.

## 6 Acknowledgments

The authors especially wish to acknowledge the assistance of their colleague Dr. J.R. Woodwark, School of Engineering, in generating the data for the pictures shown in this paper and for contributions to algorithms used in this display. The project is funded by the Science & Engineering Research Council, and their continuing support is also gratefully acknowledged.

## 7 References

1 WEBSTER, D.: 'Digital picture coding', 1978, *Wireless World*, **84**, pp. 67–70

2 DASKALAKIS, C., LAKER, R.W., and HANKINS, H.C.A.: 'The application of two dimensional compression schemes to computer graphic display systems', *Disp.*, 1981, **2**, pp. 229–235

3 HEATON, A.G., DASKALAKIS, C., MILES, P.R., and ALVI, M.A.: 'Data compression in interactive colour graphics using microprocessors', *Ibid.*, 1981, **2**, pp. 236–241
4 SCRIVENER, S.A.R.: 'The interactive manipulation of unstructured images', *Int. J. Man-Mach. Stud.*, 1982, **16**, pp. 301–313
5 EDMONDS, E.A., SCHAPPO, A., and SCRIVENER, S.A.R.: 'Image handling in two-dimensional design', *IEEE Comput. Graphics & Appl.*, 1982, **2**, pp. 75–88
6 SAMET, H.: 'Region representation: raster to quadtree conversion', CSC report 766, University of Maryland, USA, May 1979
7 HUNTER, G.M., and STEIGLITZ, K.: 'Linear transformations of pictures represented by quadtrees', *Comput. Graphics & Image Process.*, 1979, **10**, pp. 289–296
8 HUNTER, G.M., and STEIGLITZ, K.: 'Operations on images using quadtrees', *IEEE Transactions*, 1979, **PAMI-1**, pp. 145–153
9 SAMET, H.: 'Region representation: quadtrees from boundary codes', *Commun. ACM*, 1980, **23**, pp. 163–170
10 OLIVER, M.A., and WISEMAN, N.E.: 'Operations on quadtree encoded images' Comput. J., 1983, **26**, pp. 83–91
11 WOODWARK, J.R., and QUINLAN, K.M.: 'The derivation of graphics from volume models by recursive subdivision of the object space', Proc. Computer Graphics 80, 1980, pp. 335–343

**Philip Willis** is a Lecturer in computing with special interest in graphics and architectures. He has worked on hardware for flight-simulator visual systems, multiprocessors and token rings. His current graphics interests centre on very-high-resolution interactive colour paint programs and the consequent data management problems in printing applications.

**David Milford** is a Research Officer. His computing interests, which are biased towards hardware design and microprocessor applications, include graphics and speech recognition.